UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

GESTOR AVANZADO DE REQUISITOS Y PATRONES DE ATRIBUTOS DE REQUISITOS

Autor: María Marchena Ronda

Tutor: Diego Martín de Andrés

Índice de Figuras

Índice de Tablas

1 Introducción

1.1 Motivación

El proyecto de fin de carrera denominado “Gestor Avanzado de Requisitos y Patrones de Atributos de Requisitos” se desarrolla dentro del grupo de investigación Knowledge Reuse Group del departamento de Ingeniería Informática de la Universidad Carlos III de Madrid.

Para la realización de este proyecto, en el proceso de análisis y diseño, se han empleado principios básicos de modelado de la orientación a objetos empleando UML (Unified Modeling Language: Lenguaje de Modelado Unificado). Para la fase de implementación del software se ha empleado el lenguaje de programación Visual Basic .NET y el lenguaje de marcado extensible XML.

A pesar de los grandes avances tecnológicos de la ingeniería del software, existen muchos estudios [The Chaos Report 06 – BRAUDE 01] que indican que el proceso de desarrollo de software tiene aún muchas dificultades. En consecuencia, la motivación del desarrollo de este proyecto radica en la necesidad de aportar mejoras a una herramienta CASE (Computer Arded Sofware Engineering: Ingeniería del Software Asistida por Ordenador). Estas mejoras están orientadas a la personalización y gestión de los activos software (requisitos, riesgos, casos de uso, casos de prueba, etc.) que se manejarían en un proyecto específico.

1.2 Objetivos del proyecto

El objetivo del proyecto es desarrollar un software que permita la creación, en tiempo de ejecución, de plantillas para la gestión de activos software (requisitos, casos de uso, riesgos, pruebas, etc.), estas plantillas serán generadas a partir del diseño de un diagrama de elementos modelado a través de un software específico.

El diagrama de elementos es un diagrama de clases UML simplificado, cuya información relevante consta de clases con sus atributos respectivos y dos tipos de relaciones generalización y asociación. Cualquier semántica del diagrama de clases que se aleje de lo anterior citado no se tendrá en cuenta por la aplicación a realizar.

A partir de cada clase (que representa a un tipo de activo software) se generará una plantilla de trabajo donde cada atributo será una propiedad de los activos software a gestionar. Las relaciones de generalización permitirán heredar de las clases padres más atributos o propiedades para la respectiva plantilla (tipos de activos software). Por último, las relaciones de asociación permitirán representar relaciones de dependencia y/o trazabilidad entre activos software de diferentes tipos.

El software está pensado para ser usado en un proyecto de desarrollo de software. Los usuarios del presente software serían los desarrolladores de un proyecto específico. Estos usuarios serán de dos tipos: El jefe del proyecto y el analista de software. El jefe del proyecto se encargará de pensar y diseñar el diagrama de elementos y chequear las plantillas de trabajo generadas. Por su lado el analista/diseñador de software se encargará de gestionar activos por medio de las plantillas que se han generado a partir de cada elemento del diagrama diseñado por el jefe de proyecto.

Por ejemplo, al usar el software, el jefe de proyecto podrá diseñar un diagrama de elementos con la clase *requisitos* (entre otras) con los atributos *identificador, descripción y necesidad.* Luego el jefe de proyecto generará las platillas de trabajo: la clase *requisitos* se convertirá en una plantilla para gestionar *requisitos* que contendrá a su vez los campos o atributos de la clase respectiva (identificador, descripción y necesidad). A continuación, el analista podrá acceder a esta plantilla e insertar, eliminar o modificar los datos de cada requisito (activo software). Es necesario guardar una copia de estas plantillas, para su uso en futuras situaciones por parte del analista de software, que inicialmente trato con ellas o de futuros analistas que puedan reutilizarlas en otros proyectos similares.

El software desarrollado como un módulo en el presente proyecto se podrá integrar como parte de la herramienta CASE de SwReuser. Esta integración aportará una mejora en cuanto a la gestión de activos software en las diferentes fases del proceso de desarrollo de software de un proyecto específico.

1.3 Metodología

Para el desarrollo del proyecto ha sido necesario el estudio de libros de texto, distintos estándares, técnicas y herramientas relacionadas con la Ingeniería del Software, Así tenemos:

- Definiciones de la ingeniería del software.
- Las fases del proceso de desarrollo de software.
- Recursos humanos en el proceso de desarrollo de software.

Así mismo, se realizó un estudio de herramientas CASE, donde destacamos el funcionamiento y uso de la herramienta SwReuser como base en la que se va a asentar el proyecto.

Otro punto a destacar, es el aprendizaje del lenguaje unificado de modelado (UML Unified Modeling Language). Se abarcó tanto los conceptos básicos como el estudio intensivo de los principales diagramas. Este estudio fue necesario para la especificación de las fases del proceso de desarrollo del software, motivo del presente proyecto.

Estudio del lenguaje de programación *Visual Basic .NET*. Este lenguaje fue escogido por sus principales bondades y principalmente porque facilitaría la futura integración con la herramienta SwReuser.

Estudio del lenguaje *XML* (*Extensible Markup Language*).Este lenguaje fue escogido para guardar las plantillas generadas con Visual Basic .NET en un formato XML y a la vez para recuperar dichos archivos y transformarlos en las plantillas editables.

A continuación, se desarrollaron las fases del proceso de desarrollo del software motivo del presente proyecto: análisis, diseño, implementación y pruebas.

1.4 Contenido de la memoria

Para especificar el proyecto en curso dividimos este documento en las siguientes partes generales:

En la primera parte del documento (**Introducción**, **Estado del Arte** y **Tecnología Utilizada**) especificamos la visión general del proyecto (el software a realizar), los conocimientos generales orientados a la mejor comprensión del proyecto por parte de los lectores y las herramientas utilizadas para el desarrollo del proyecto en cuestión.

El apartado dedicado a los **objetivos** del proyecto se centra en los casos de usos del sistema, los requisitos de los usuarios tanto funcionales y de restricción y por último los riesgos que plantea el proyecto.

Los puntos de **análisis y diseño** recogerán todas las especificaciones o requisitos necesarios que definen los servicios que debe brindar el software a desarrollar y la especificación de todas las características técnicas detalladas, que definan como será la aplicación que satisfaga los requisitos especificados.

El **presupuesto del proyecto** contiene la planificación inicial y final del mismo y de los costes que suponen el desarrollo del proyecto.

El **plan de pruebas** pensadas para la aplicación, es un referente importante para conocer los objetivos alcanzados por el software diseñado. De este modo, se conocen los posibles fallos o errores del sistema así como el logro de los resultados correctos.

El **manual de usuario,** redactado para ayudar al usuario en el manejo de la aplicación. De este modo, el usuario podrá consultar sus posibles dudas y en pocas hojas tener una idea bastante clara de cómo utilizar dicha aplicación.

Por último, las **conclusiones** y **trabajos futuros** incluirán todas las reflexiones finales que nos aporta el desarrollo del proyecto y posibles trabajos futuros que lo puedan complementar y/o mejorar.

1.5 Acrónimos y definiciones

PC: Personal Computer. Es la expresión estándar que se utiliza para denominar a las computadoras personales en general.

UML: Unifies Modeling Language (lenguaje de modelado unificado). Lenguaje estándar que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos.

SwREUSER: Aplicación software basada en una herramienta computer-aided software enginereing (CASE) diseñada para facilitar el desarrollo de sistemas de información basados en el paradigma de la reutilización.

CAKE: Computer Aided Knowledge Environment, es un entorno de herramientas de aplicaciones y mitología para la identificación, clasificación, recuperación, organización y reutilización de conocimiento.

Diagrama de elementos: Diagrama de clases que crea el jefe de proyecto para especificar que activos software se gestionan en un proyecto específico.

Elemento: es denominado a cada una de las clases que se encuentran incluidas dentro del diagrama de elementos, anteriormente citado.

Plantilla de trabajo: Elemento de información que recoge cada uno de los atributos contenidos por la clase dentro del diagrama de elementos generada por el jefe de proyecto. Esta plantilla servirá para la gestión del activo software respectivo por los analistas o diseñadores.

Activos software: Subproductos del producto software, es decir, se considera a cada uno de los componentes en los que se puede dividir el producto software final.

$\sum$(Activos software) = Software final.

VB.net: Lenguaje de programación Visual Basic .NET.

Datagridview: El control DataGridView proporciona una forma eficaz y flexible de mostrar datos en formato de tabla. Puede utilizar el control DataGridView para mostrar vistas de sólo lectura de una cantidad pequeña de datos o puede ajustar su tamaño para mostrar vistas modificables de conjuntos muy grandes de datos.

Listview: es parecido al control Listbox excepto que puede mostrar sus elementos de muchas formas, junto con cualquier número de subelementos por cada elemento.

XML: (Extensible Markup Language/Lenguaje de marcas extensible) es un metalenguaje extensible de etiquetas.

IEEE: (The Institute of Electrical and Electronics Engineers/el Instituto de Ingenieros Eléctricos y Electrónicos) es una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

LAN: Local Area Network. Es una red de área local.

Diagrama Gantt: es una herramienta gráfica, cuyo objetivo es el de mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. A pesar de que, en principio, el diagrama de Gantt no indica las relaciones existentes entre actividades, la posición de cada tarea a lo largo del tiempo hace que se puedan identificar dichas relaciones e interdependencias.

Patrones de Diseño: son la base para la búsqueda de soluciones a problemas comunes en el desarrollo del software y otros ámbitos referentes al diseño de interacción o interfaces.

2 Estado del arte

2.1 Ingeniería del Software

2.1.1 Conocimientos básicos de Ingeniería del Software

La ingeniería del software es la rama de la ingeniería que crea y mantiene las aplicaciones de software aplicando tecnologías y prácticas de las ciencias computacionales, manejo de proyectos, ingeniería, el ámbito de la aplicación, y otros campos.

La ingeniería del software, como las disciplinas tradicionales de ingeniería, tiene que ver con el costo y la confiabilidad. Algunas aplicaciones de software contienen millones de líneas de código que se espera que se desempeñen bien en condiciones siempre cambiantes.

La ingeniería del software es un término que fue utilizado por primera vez en 1968 por Fritz Bauer en la primera conferencia sobre el desarrollo de software. Se define por Alan Davis como “la aplicación inteligente de principios probados, técnicas, lenguajes y herramientas para la creación y mantenimiento, dentro de un coste razonable, de software que satisfaga las necesidades de los usuarios”

Según la definición del IEEE, “software es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo”. El mismo autor, Lewis, decía “un producto de software es un producto diseñado para un usuario”.

En este contexto, la ingeniería de software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software. Es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas a los problemas de desarrollo del software, por ello permite elaborar de forma consistente productos correctos y utilizables.

El proceso de ingeniería del software es “un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, la obtención de un producto de software de calidad”. El proceso de desarrollo del software es “aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño, el diseño implementado en código, el código es probado, documentado y certificado para un uso operativo“. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo.

El concepto de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le llama el ciclo de vida del software que comprende cuatro grandes fases: concepción, elaboración, construcción y transición.

- Concepción: define el alcance del proyecto y desarrolla un caso de negocio.
- Elaboración: define un plan de proyecto y desarrolla un caso de negocio. La elaboración define un plan del proyecto, especifica las características y fundamenta la arquitectura.
- Construcción: es la creación el producto.
- Transición: transfiere el producto a los usuarios.

Actualmente el enfoque Orientado a Objetos (OO) se encuentra en una etapa de madurez como paradigma del desarrollo de sistemas de información. El Object Management Group (OMG) es un consorcio a nivel internacional que integra a los principales representantes de la industria de la tecnología de información OO. Su objetivo principal es la promoción, fortalecimiento e impulso de la industria OO. El OMG propone y adopta por consenso especificaciones entorno a la tecnología OO. Una de las especificaciones más importantes es la adopción en 1998 del Lenguaje de Modelado Unificado o UML (del inglés Unified Modeling Language) como un estándar, que junto con el Proceso Unificado están consolidando la tecnología OO.

El paradigma de lo Orientado a Objetos

Antes de analizar los pasos del proceso de desarrollo de software se expondrán los conceptos fundamentales del paradigma que guía la tecnología OO.

Existen conceptos ligados en torno a la tecnología orientada a objetos: el paradigma, los principios, el análisis y el diseño, mismos que a continuación se comentan.

La Programación Orientada a Objetos

La programación orientada a objetos como paradigma “es una forma de pensar, una filosofía, de la cual surge una cultura nueva que incorpora técnicas y metodologías diferentes. Pero estas técnicas y metodologías, y la cultura misma, provienen del paradigma, no lo hacen. La OOP como paradigma es una postura ontológica: el universo computacional está poblado por objetos, cada uno responsabilizándose por sí mismo, y comunicándose con los demás por medio de mensajes”.

Se debe distinguir que la OOP como paradigma y como metodología no son la misma cosa. Sin embargo, la publicidad confunde asociando la OOP más una metodología, que al paradigma. De aquí que “el interés en la OOP radica más en los mecanismos que aporta para la construcción de programas que en aprovechar un esquema alterno para el modelado de procesos computacionales”.

La Programación Orientada a Objetos desde el punto de vista computacional “es un método de implementación en el cuál los programas son organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases, todos son miembros de una jerarquía de clases unidas vía relaciones de herencia”.

Fundamentos del desarrollo de aplicaciones

A continuación se van a describir los conceptos necesarios para acometer la realización de este proyecto, estos conceptos incluyen los principios de la programación orientada a objetos, la metodología que se ha seguido en la elaboración del proyecto y una introducción al lenguaje de modelado UML.

Fundamentos de la Programación Orientada al Objeto

El término orientado a objetos significa que el software se construye como una colección de objetos discretos que engloban tanto estructuras de datos como su comportamiento. La diferencia respecto a la programación convencional radica en que la relación entre las estructuras de datos y su comportamiento es mucho más débil que en la orientación a objetos.

Características de los objetos

Se define *objeto* como una entidad delimitada con precisión e identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento por sus operaciones y métodos.

Cada objeto tiene una identidad propia que le diferencia de los demás, incluso de los objetos con idéntica estructura y mismos valores de sus atributos. La identidad es la denominación gracias a la cual se puede hacer referencia a un objeto de modo exclusivo.

Clasificación significa que los objetos con la misma estructura de datos y el mismo comportamiento se unen para formar una clase de objetos. Una *clase* es una abstracción que describe las propiedades relevantes para una aplicación y que ignora las demás. La selección de clases es arbitraria y depende de la aplicación.

Una clase describe un conjunto infinito de objetos individuales. Se dice que cada objeto es una instancia de su clase. Toda instancia de la clase posee sus propios valores para sus atributos, pero comparte los nombres de estos y las operaciones con las demás instancias de la clase.

Polimorfismo significa que una misma operación puede comportarse de modos distintos en clases diferentes. Una operación es una acción o transformación que se lleva a cabo o se puede aplicar a un objeto. La operación específica de una cierta clase se denomina método. Dado que los operadores en la orientación a objetos son polimórficos, puede haber más de un método que los implemente.

Con frecuencia un conjunto de abstracciones forma una jerarquía. La identificación de estas jerarquías en el diseño simplifica la comprensión del problema.

La herencia es compartir atributos y operaciones entre clases tomando como base una relación jerárquica. En términos generales se puede definir una clase que después se irá refinando para producir subclases. Todas las subclases poseen o heredan todas las propiedades de su superclase y añaden además otras que le son propias y le distinguen de las otras subclases. La capacidad de factorizar el comportamiento común de las propiedades de varias clases en una superclase común y de heredar las propiedades de la superclase reduce la repetición en el diseño y en la implementación de los programas, siendo una de las principales ventajas de la programación orientada a objetos.

Elementos fundamentales de la orientación a objetos

ABSTRACCIÓN

La abstracción consiste en centrarse en los aspectos esenciales inherentes de una entidad e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y hace un objeto antes de decidir cómo debería implementarse. La abstracción surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real y la decisión de concentrarse en las similitudes e ignorar por el momento las diferencias.

Una abstracción se centra en la visión externa de un objeto, sirviendo para separar el comportamiento esencial de un objeto de su implementación. El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño.

Un uso adecuado de la abstracción, permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de datos y documentación.

ENCAPSULAMIENTO

Es una técnica de modelado e implementación que separa los aspectos externos de un objeto de los internos, detalles de implementación de un objeto.

El encapsulamiento evita que el programa llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin que afecte a las aplicaciones que lo utilizan.

MODULARIDAD

La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y, débilmente acoplados.

Este consiste en dividir un programa en módulos que pueden compilarse de forma separada pero, que tienen conexiones con otros módulos. Las conexiones entre módulos son las suposiciones que cada módulo hace acerca de los demás.

La fragmentación de un programa en componentes individuales, tiene por ventaja reducir la complejidad y, crear una serie de fronteras bien definidas y documentadas en el programa.

MENSAJES

Mensaje es una llamada a una operación o a un objeto, en la que se incluye el nombre de la operación y una lista de valores de argumentos. Es una comunicación entre objetos que transmite información con la expectativa de desatar una acción, la recepción de un mensaje se considera como un evento.

Un objeto accede a otro enviándole un mensaje, cuando esto ocurre el receptor ejecuta el método correspondiente al mensaje.

El método asociado a un mensaje es el algoritmo detallado que lo implementa que es privado. El conjunto de mensajes a los que un objeto responde caracteriza su comportamiento.

Elementos secundarios de la orientación a objetos

TIPOS

Los tipos son el reforzamiento de clases, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse de formas muy restringidas.

Se incluye el tipo como elemento separado de modelado de objetos porque el concepto de tipo pone énfasis en el significado de la abstracción en un sentido muy distinto.

POLIMORFISMO

Es la propiedad por la que una operación se comporta de forma diferente en diferentes clases. De esta manera un mensaje será interpretado de maneras distintas según el objeto que lo recibe.

Esta técnica permite que una variable o función adopte diferentes formas en tiempo de ejecución o, más específicamente, posibilidad de referirse a instancias de varias clases.

Un solo nombre puede denotar objetos de muchas clases diferentes que se relacionan por alguna superclase común. La faceta más interesante del polimorfismo aparece cuando interactúan las características de la herencia y la ligadura dinámica. Implica que el objeto que envía un mensaje no precisa conocer la instancia de la clase receptora.

El polimorfismo permite detectar y aprovechar similitudes entre distintas clases de objetos, objetos distintos que responden a un mismo mensaje pueden ser tratados de la misma forma por el remitente.

CONCURRENCIA

La concurrencia es la propiedad que distingue a un objeto activo de uno que no está activo. Permite a diferentes objetos actuar al mismo tiempo.

La concurrencia precisa de la existencia de un sistema distribuido de procesadores o, un sistema multitarea. La concurrencia se centra en la abstracción de procesos y la sincronización de estos. Un sistema que implique concurrencia puede tener muchos hilos de control, algunos transitorios y otros permanentes.

PERSISTENCIA

Es la propiedad de un objeto por la que su existencia trasciende el tiempo es decir, el objeto continúa existiendo tras la destrucción de sus creados, o el espacio es decir, la posición del objeto varía con respecto al espacio de direcciones en que fue creado.

Metodologías

En cuanto a las metodologías OO, hay un gran número de métodos orientado a objetos actualmente. Muchos de los métodos pueden ser clasificados como orientados a objetos porque soportan de manera central los conceptos de la orientación a objetos. Algunas de las metodologías más conocidas y estudiadas hasta antes del UML son:

- Object-Oriented Design (OOD), Booch.
- Object Modeling Technique (OMT), Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Hierarchical Object Oriented Design (HOOD), ESA.
- Object Oriented Structured Design (OOSD), Wasserman.
- Object Oriented Analysis (OOSA), Shaler y Mellor.
- Responsability Driven Design (RDD), Wirfs-Brock, entre otros.

Actualmente las metodologías más importantes de análisis y diseño de sistemas han confluido en lo que es el UML, bajo el respaldo del Object Management Group.

2.1.2 Recursos humanos para la Ingeniería del Software

La constitución del equipo de trabajo es la actividad más delicada a la que se enfrenta un jefe de proyecto y en la que más debe demostrar sus capacidades. El equipo es creado para una operación determinada y está compuesto por personas sobre las que no existe normalmente un poder jerárquico, provenientes de diversos departamentos o especialidades, y que ha de funcionar como un todo armónico y ser capaz de conseguir los resultados esperados que, por definición, son complejos, inusuales y arriesgados.

Cada una de las personas que conforman este equipo de trabajo posee un perfil, el cual le caracteriza en un tipo de actividad. No todos los perfiles son necesarios durante todo el proyecto ni en todos los proyectos. En función, del ciclo de vida empleado y de las actividades a realizar, se pueden determinar a priori los perfiles requeridos.

- Para la asignación de un perfil, es necesario tener en cuenta los siguientes aspectos:
- Conocimientos generales requeridos.
- Conocimientos técnicos especializados requeridos.
- Habilidades de comunicación requeridas.
- Actitudes requeridas en el trabajo.
- Relación con otros perfiles.
- Recursos materiales asociados al perfil.
- Características temporales.

Para el presente proyecto, los usuarios para los que está destinada esta herramienta son el jefe de proyecto y el analista/diseñador de software, aunque el resto del equipo de trabajo puede utilizarla según el reparto de funciones dentro del mismo equipo, cuestión ajena al desarrollo de la aplicación.

El jefe de proyecto destaca como la figura clave en la planificación, ejecución y control del proyecto y es el motor que ha de impulsar el avance del mismo mediante la toma de decisiones tendentes a la consecución de los objetivos. El jefe de proyecto tiene poder ejecutivo, aunque no tiene un poder absoluto, ya que se encuentra inmerso en la estructura y organización de la empresa.

Las relaciones básicas del jefe de proyecto con otras unidades o personas dependen, en gran medida, de la estructura organizativa que posea la organización. A continuación, se muestra el caso de una empresa que sigue una estructura orientada a proyectos, donde se observa la importancia del jefe de proyecto.

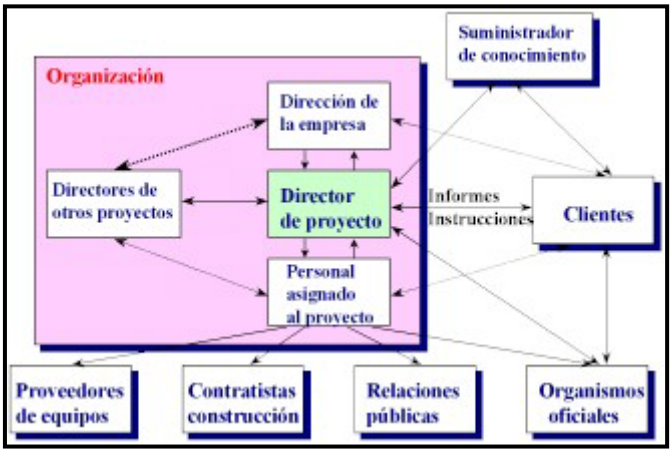


Ilustración 1: Organización de proyecto.

A partir del diagrama expuesto, se puede sacar en claro cuáles son las funciones del jefe de proyecto. Su misión podría resumirse en dirigir el equipo del que dispone para alcanzar los objetivos del proyecto. Más concretamente se podría destacar las siguientes funciones específicas.

- Colaboración con el cliente en la definición de los objetivos del proyecto.
- Planificación del proyecto en todos sus aspectos, identificando las actividades a realizar, los recursos a poner en juego, los plazos y los costes previstos.
- Dirección y coordinación de todos los recursos empleados en el proyecto.
- Mantenimiento permanente de las relaciones externas del proyecto: clientes, proveedores, subcontratistas, otras direcciones, etc.
- Toma de decisiones necesarias para conocer en todo momento la situación en relación con los objetivos establecidos.
- Adopción de las medidas correctoras pertinentes para poner remedio a las desviaciones que se hubieran detectado.
- Responder ante clientes y superiores de la consecución de los objetivos del proyecto.
- Proponer, en su caso, modificaciones a los límites u objetivos básicos del proyecto cuando concurran circunstancias que así lo aconsejen.

Esta definición de funciones no puede considerarse exhaustiva. En cada entidad sería necesario hacer una definición de funciones más concreta y adaptada a las características particulares de cada proyecto.

Por otro lado, el analista software se caracteriza por su capacidad de abstracción ya que ejerce las tareas de análisis de los sistemas informáticos, con el fin de automatizarlos. El analista tiene por cometido estudiar en profundidad un problema y describirlo con el propósito de ser solucionado mediante un sistema informático, esta idea será diseñada por el diseñador software e implementada por el programador. Con todo, en la actualidad estas funciones han quedado un tanto obsoletas debido a los avances de la ingeniería del software, estas no serían suficientes para lograr un mínimo éxito en el desarrollo de software.

Funciones tales como la dirección de recursos hacia el resultado deseado, o el estudio de requisitos para determinar el comportamiento que se espera del software, la garantía de calidad, para garantizar las expectativas del cliente, un boceto de lo que podría ser el diseño para que exista una mínima certeza de que el software es viable y eficaz con la tecnología existente y la gestión de la configuración para controlar el posible caos a medida que el software crece.

También es deseable que el analista tenga conocimientos básicos de usabilidad, ya que cualquier sistema que no esté al servicio de los usuarios, no tiene mucho sentido.

En puntos siguientes, se nombrarán a ambos usuarios que utilizarán la herramienta y se especificarán en detalle su uso de la aplicación.

2.2 UML

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (principalmente Booch, OMT y OOSE). UML ha puesto fin a las llamadas "guerras de métodos" que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

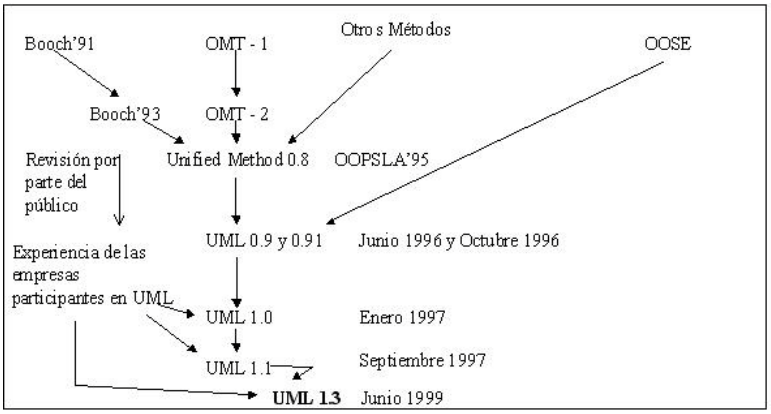


Ilustración 2: Evolución UML

Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. En la Figura se puede ver cuál ha sido la evolución de UML hasta la creación de UML 1.3. Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación.

Diagramas

En UML 2.0 hay 13 tipos diferentes de diagramas. Para su comprensión, es útil categorizarlos jerárquicamente como se muestra en la figura de a continuación:

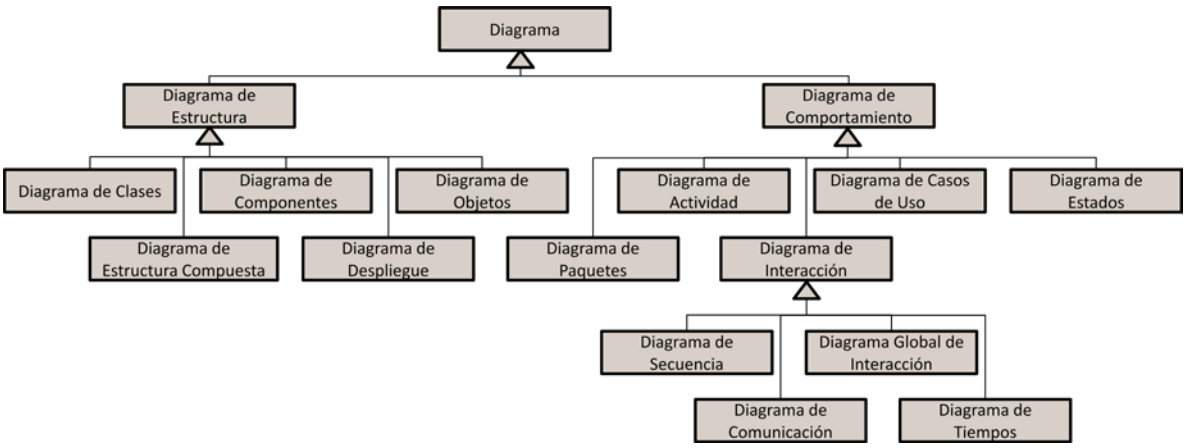


Ilustración 3: Diagramas en UML 2.0

La importancia de cada uno de estos diagramas es que al igual que en dibujo técnico donde la planta y el alzado de una figura nos muestra la misma figura vista desde distintos ángulos, cada diagrama nos permite fijarnos en un aspecto distinto del sistema.

Los **diagramas de estructura** enfatizan en los elementos que deben existir en el sistema de modelado.

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta
- Diagrama de despliegue
- Diagrama de paquetes

Los **diagramas de comportamiento** enfatizan en lo que debe suceder en el sistema de modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de estados

Los **diagramas de interacción** son un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia
- Diagrama de comunicación (que es una versión simplificada del Diagrama de colaboración)
- Diagrama de tiempos
- Diagrama global de interacciones o Diagrama de vista de interacción

No es necesario utilizar cada una de las herramientas anteriores, solo aquellas que sean más adecuadas a las necesidades planteadas, para ello es útil conocer la utilidad y las limitaciones de cada una de ellas.

Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema, mostrando sus clases, atributos y relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Aspectos importantes de las clases:

Propiedades también llamados atributos o características, son valores que corresponden a un objeto. Generalmente se conoce como la información detallada del objeto.

Operaciones son aquellas actividades o verbos que se pueden realizar con/para este objeto.

Interfaz es un conjunto de operaciones y/o propiedades que permiten a un objeto comportarse de cierta manera, por lo que define los requerimientos mínimos del objeto.

Herencia se define como la reutilización de un objeto padre ya definido para poder extender la funcionalidad en un objeto hijo. Los objetos hijos heredan todas las operaciones y/o propiedades de un objeto padre.

Las clases pueden estar asociadas de diferentes maneras:

Generalización.

La herencia es uno de los conceptos fundamentales de la programación orientada a objetos, en la que una clase toma todos los atributos y operaciones de la clase de la que es heredera, y puede alterar/modificar algunos de ellos, así como añadir más atributos y operaciones propias.

En UML, una asociación de generalización entre dos clases, coloca a estas en una jerarquía que representa el concepto de herencia de una clase derivada de la clase base.

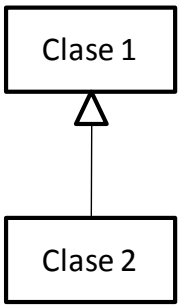


Ilustración 4: Generalización

Asociaciones.

Una asociación representa una relación entre clases, y aporta la semántica común y la estructura de muchos tipos de “conexiones” entre objetos.

Las asociaciones son los mecanismos que permiten a los objetos a comunicarse entre sí. Describe la conexión entre diferentes clases (la conexión entre los objetos reales se denomina conexión de objetos o enlace).

Las asociaciones pueden tener un papel que especifica el propósito de la asociación y puede ser unidireccionales o bidireccionales (indicando si los dos objetos participantes en la relación pueden intercambiar mensajes entre sí, o es únicamente uno de ellos el que recibe información del otro). Cada extremo de la asociación también tiene un valor de multiplicidad, que indica cuántos objetos de ese lado de la asociación están relacionados con un objeto del extremo contrario.

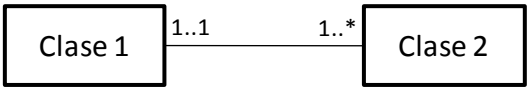


Ilustración 5: Asociación

Agregación.

La agregación es un tipo especial de asociación en las que las dos clases participantes no tienen un estado igual, pero constituyen una relación «completa». Una agregación describe cómo se compone la clase que asume el rol completo de otras clases que se encargan de las partes. En la agregación, la clase que actúa como completa, tiene una multiplicidad de uno.

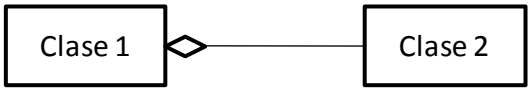


Ilustración 6: Agregación

Composición.

Las composiciones son asociaciones que representan acumulaciones muy fuertes. Esto significa que las composiciones también forman relaciones completas, pero dichas relaciones son tan fuertes que las partes no pueden existir por sí mismas. Únicamente existen como parte del conjunto, y si este es destruido las partes también lo son.

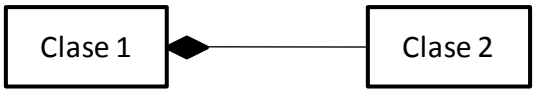


Ilustración 7: Composición

Diagrama de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Debido a que estos son más parecidos a los diagramas de casos de usos estos son utilizados para modelar la vista estática y dinámica de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema.

En él se situarán librerías, tablas, archivos, ejecutables y documentos que formen parte del sistema.

Uno de los usos principales es que puede servir para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

Diagrama de objetos

Los diagramas de objetos son utilizados durante el proceso de Análisis y Diseño de los sistemas informáticos en la metodología UML.

Se puede considerar un caso especial de un diagrama de clases en el que se muestran instancias específicas de clases (objetos) en un momento particular del sistema. Los diagramas de objetos utilizan un subconjunto de los elementos de un diagrama de clase. Los diagramas de objetos no muestran la multiplicidad ni los roles, aunque su notación es similar a los diagramas de clase.

Diagrama de estructura compuesta

Un diagrama de estructura compuesta es un tipo de diagrama de estructura estática en el Lenguaje de Modelado Unificado (UML), que muestra la estructura interna de una clase y las colaboraciones que esta estructura hace posibles. Esto puede incluir partes internas, puertas mediante las cuales, las partes interactúan con cada una de las otras o mediante las cuales, instancias de la clase interactúan con las partes y con el mundo exterior, y *conectores* entre partes o puertas. Una *estructura compuesta* es un conjunto de elementos interconectados que colaboran en tiempo de ejecución para lograr algún propósito. Cada elemento tiene algún *rol* definido en la colaboración.

Diagrama de despliegue

El Diagrama de Despliegue es un tipo de diagrama que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.

Los elementos usados por este tipo de diagrama son nodos (representados como un prisma), componentes (representados como una caja rectangular con dos protuberancias del lado izquierdo) y asociaciones.

En el UML 2.0 los componentes ya no están dentro de nodos. En cambio, puede haber artefactos u otros nodos dentro de un nodo. Un artefacto puede ser algo como un archivo, un programa, una biblioteca, o una base de datos construida o modificada en un proyecto. Estos artefactos implementan colecciones de componentes. Los nodos internos indican ambientes, un concepto más amplio que el hardware propiamente dicho, ya que un ambiente puede incluir al lenguaje de programación, a un sistema operativo, un ordenador o un clúster de terminales.

La mayoría de las veces el modelado de la vista de despliegue implica modelar la topología del hardware sobre el que se ejecuta el sistema. Aunque UML no es un lenguaje de especificación hardware de propósito general, se ha diseñado para modelar muchos de los aspectos hardware de un sistema a un nivel suficiente para que un ingeniero software pueda especificar la plataforma sobre la que se ejecuta el software del sistema.

El uso principal de este tipo de diagramas se basa en sistemas empotrados, sistemas cliente-servidor y sistemas completamente distribuidos.

Diagrama de paquetes

El diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.

Diagrama de actividades

El diagrama de actividades representa los flujos de trabajo paso a paso de negocio y operacionales de los componentes en un sistema. Un Diagrama de Actividades muestra el flujo de control general.

Diagrama de casos de uso

El Diagrama de Casos de Uso define una notación gráfica que representa la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea.

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones.

Un actor es algo con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema.

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad. Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación del caso de uso. Para el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tres tipos:

Include (<<>>): Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso.

Extend (<<>>): Un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación include que se da siempre que se realiza la interacción descrita)

Los casos de uso no son parte del diseño (cómo), sino parte del análisis (qué), de forma que nos ayudan a describir lo que el sistema debe hacer. Los casos de uso son qué hace el sistema desde el punto de vista del usuario, es decir, describen un uso del sistema y cómo este interactúa con el usuario.

Con todo el diagrama de casos de uso no es lo realmente importante, ya que lo útil es el documento que describe el caso de uso, donde se detalla la forma de interactuar entre el sistema y el usuario.

Nombre:	Nombre del caso de uso
Autor:	Autor del caso de uso
Fecha:	Fecha de creación
Descripción:	
Descripción corta del caso de uso	
Actores:	
Actores del sistema que actúan en el caso de uso	
Precondiciones:	
Condiciones previas para darse el caso de uso	
Flujo Normal:	
Descripción detallada del caso de uso	
Flujo Alternativo:	
Si no se cumplen las precondiciones	
Poscondiciones:	
Descripción del caso de uso una vez acabado su flujo normal	

Tabla 1: Descripción de un caso de uso

Saltándose los campos evidentes como nombre, autor, fecha y descripción; los actores son aquellos que interactúan con el sistema. Las precondiciones son los hechos que se han de cumplir para que el flujo de evento se pueda llevar a cabo. Luego tenemos el flujo de eventos, que corresponde con la ejecución normal y exitosa del caso de uso (use case). Los flujos alternativos son los que nos permiten indicar qué es lo que hace el sistema en los casos menos frecuentes e inesperados. Por último, las poscondiciones son los hechos que se han de cumplir si el flujo de eventos normal se ha ejecutado correctamente. De forma que un caso de uso (use case) es un documento como el anteriormente presentado. Los casos de uso se pueden detallar más o menos dependiendo de la necesidad del problema.

Diagrama de estados

Los diagramas de estado muestran los diferentes estados de un objeto durante su vida, y los estímulos que provocan los cambios de estado en un objeto.

Los diagramas de estado ven a los objetos como *máquinas de estado* o autómatas finitos que pueden estar en un conjunto de estados finitos y que pueden cambiar su estado a través de un estímulo perteneciente a un conjunto finito.

De ese modo, los estados son considerados los ladrillos de los diagramas de estado. Un estado pertenece a exactamente una clase y representa un resumen de los valores y atributos que puede tener la clase. Un estado UML describe el estado interno de un objeto de una clase particular

No todos los cambios en los atributos de un objeto deben estar representados por estados, sino únicamente aquellos cambios que pueden afectar significativamente a la forma de funcionamiento del objeto.

Diagrama de secuencia

El diagrama de secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Mientras que el diagrama de casos de uso permite el modelado de una vista business del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. Típicamente uno examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si tienes modelada la descripción de cada caso de uso como una secuencia de varios pasos, entonces puedes "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos.

Existen dos tipos de mensajes: síncronos y asíncronos. Los mensajes síncronos se corresponden con llamadas a métodos del objeto que recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensajes se representan con flechas con la cabeza llena. Los mensajes asíncronos terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia. Se representan con flechas con la cabeza abierta.

Diagrama de comunicación

El diagrama de comunicación es una versión simplificada del diagrama de colaboración de la versión de UML 1.x.

Un diagrama de Comunicación modela las interacciones entre objetos o partes en términos de mensajes en secuencia. Los diagramas de Comunicación representan una combinación de información tomada desde el diagrama de Clases, Secuencia, y Diagrama de casos de uso describiendo tanto la estructura estática como el comportamiento dinámico de un sistema.

Los diagramas de comunicación y de secuencia describen información similar, y con ciertas transformaciones, pueden ser transformados unos en otros sin dificultad.

Diagrama de tiempos

Los diagramas de tiempo son una representación especial de interacción que se enfoca en el tiempo de los mensajes enviados entre objetos. Se pueden usar estos diagramas para mostrar restricciones detalladas sobre el tiempo, ó para mostrar los cambios con líneas de vida respecto al tiempo. Los diagramas de tiempo son generalmente utilizados con sistemas en tiempo real o en sistemas embebidos.

Diagrama global de interacciones o Diagrama de vista de interacción

El diagrama global de las interacciones es un *diagrama de comportamiento*, más precisamente, uno de los cuatro *diagramas de interacción*. Muestra una cierta vista sobre los aspectos dinámicos de los sistemas modelados. Aunque un diagrama global de las interacciones es una representación gráfica de una interacción, éste se distingue fuertemente de los diagramas de secuencia y de comunicación, dos de los otros diagramas de interacción. De hecho, algunos elementos gráficos del diagrama global de las interacciones están tomados del diagrama de actividades, otro diagrama de comportamiento para el modelado de actividades.

Los modelos de interacción pueden llegar a ser muy grandes para sistemas complejos. Si el número de líneas de vida participantes y el número de mensajes intercambiados exceden una cierta medida, se impone “modularizar” las interacciones y dividir en partes pequeñas, más manejables, de acuerdo a principios universales del diseño de sistemas, que también pueden ser visualizadas con la ayuda de un clásico diagrama de secuencias.

2.3 Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo del software reduciendo el coste de las mismas en términos de tiempo y dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Los objetivos de toda herramienta CASE son:

1. Mejorar la productividad en el desarrollo y mantenimiento del software.
2. Aumentar la calidad del software.
3. Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la planificación de un proyecto.
5. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. Automatizar, el desarrollo del software, documentación, generación de código, pruebas de errores y gestión de proyecto.
7. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
8. Gestión global en todas las fases de desarrollo del software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Clasificación de herramientas CASE

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

1. Las plataformas que soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que cubren.
3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

La siguiente clasificación es la más habitual basada en las fases del ciclo de desarrollo que cubren:

- *Upper CASE (U-CASE)*, herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.
- *Middle CASE (M-CASE)*, herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- *Lower CASE (L-CASE)*, herramientas que semiautomatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de Desarrollo rápido de aplicaciones.

Existen otros nombres que se le dan a este tipo de herramientas, y que no es una clasificación excluyente entre sí, ni con la anterior:

- *Integrated CASE (I-CASE)*, herramientas que engloban todo el proceso de desarrollo software, desde análisis hasta implementación.
- *MetaCASE*, herramientas que permiten la definición de nuestra propia técnica de modelado, los elementos permitidos del metamodelo generado se guardan en un repositorio y pueden ser usados por otros analistas, es decir, es como si definiéramos nuestro propio UML, con nuestros elementos, restricciones y relaciones posibles.
- *CAST (Computer-Aided Software Testing)*, herramientas de soporte a la prueba de software.
- *IPSE (Integrated Programming Support Environment)*, herramientas que soportan todo el ciclo de vida, incluyen componentes para la gestión de proyectos y gestión de la configuración.

Por funcionalidad podríamos diferenciar algunas como:

- Herramientas de generación semiautomática de código.
- Editores UML.
- Herramientas de Refactorización de código.
- Herramientas de mantenimiento como los sistemas de control de versiones·

2.3.1 SwREUSER

SOFTWARE REUSE COMPANY.

La ingeniería del software es importante a la hora de acometer proyectos de desarrollo de software, para ello es necesario considerar dos necesidades clave:

- Disponer de un buen proceso.
- Tener como base buenas herramientas acordes al proceso que se va a realizar.

Digamos que ambas se complementas, es necesario tanto un buen proceso como disponer de las herramientas adecuadas.

La aplicación utilizada “The reuse company” responde a estas necesidades, aportando tanto el proceso (La metodología IRM – Incremental Reuse Method) como un conjunto de herramientas CASE.

Dado a que cumple con las especificaciones de la metodología IRM, es capaz de conectarse con el repositorio Reuse Server y almacenar allí todos los activos gestionados durante el ciclo de vida. Asimismo, y gracias al soporte a la fase de debriefing (post-mortem) definida en IRM, permite que la reutilización de activos sea siempre lo efectiva que todo proyecto necesita.

En línea con el enfoque de The Reuse Company, que aportan a las organizaciones los medios para que puedan reutilizar su conocimiento, IRM y el conjunto de herramientas CASE están enfocados hacia la reutilización de software.

SwREUSER, es la aplicación software sobre la que se va a montar el proyecto dado, esta se basa en una herramienta computer-aided software engineering (CASE) diseñada para permitir a los ingenieros de software el desarrollo de sistemas de información basados en el paradigma de la Reutilización, utilizando la gestión del conocimiento apropiada. Por lo tanto, dicho software maneja un tipo más de representación de conocimiento. Para poder manejar este tipo de representación dentro de una herramienta CASE, en el que el conocimiento es el activo principal para el desarrollo de sistemas de información.

CAKE Computer Aided Knowledge Environment, es un entorno de herramientas, aplicaciones y metodologías para la identificación, clasificación, recuperación, organización y reutilización de conocimiento. El entorno CAKE ha sido diseñado para gestionar, organizar y reutilizar todo tipo de 'activo de conocimiento' generado. La gestión de conocimiento de CAKE, así como su entorno de ingeniería están basados en un esquema de representación y clasificación denominado RSHP.

CAKE mezcla Ingeniería de la Información, Ingeniería de Negocio (inteligencia), Ingeniería del Software, Ingeniería de Dominios y la Gestión del Conocimiento clásica en una misma disciplina, la Ciencia del Conocimiento. Así pues, el objetivo principal de CAKE se centra en ayudar a toda la organización en su intención de afrontar y dominar esta Ciencia del Conocimiento.

swREUSER: Descripción general.

Como herramienta CASE, swREUSER está completamente basado en el lenguaje UML (Unified Modeling Language), manteniendo una de las representaciones de su semántica más precisas del mercado. Sin embargo, con el propósito de soportar gestión del conocimiento y reutilización dicho software permite llevar a cabo actividades de Ingeniería y Análisis de Dominios, gracias al módulo dmCAKE, que recopila una representación completa del conocimiento que la organización posee sobre un dominio dado, esto permite avanzadas técnicas de gestión dentro de la herramienta. Esto es gestionado por el software a partir de REUSEServer que permite llevar a cabo actividades de reutilización institucional de alto nivel, incluyendo la reutilización de análisis, diseños, requisitos, riesgos... Por ello el interfaz de recuperación se separa de un modelo convencional, en el cual el usuario puede crear un pequeño diagrama UML que será contrastado para devolver aquellos modelos que más se parecen a éste.

swREUSER cubre no solo las fases de Análisis y Diseño con UML, sino también otras etapas del ciclo de vida de desarrollo: gestión de requisitos, gestión de riesgos, estimación de costes y recursos, conexión con la etapa de planificación, modelado mediante UML, control de proyectos mediante diferentes métricas, generación de código, control de calidad... Además se ha implementado un potente sistema de traza entre todos estos elementos.

Con todo, la herramienta no ha olvidado la importancia de la Compartición del Conocimiento: modelos UML pueden ser compartidos, debatidos, comparados de forma automática y transformados utilizando un sistema completo de foros

Además, dispone de una metodología asociada ROM (Reuse Oriented Method) cubre la siguiente funcionalidad: Alta precisión semántica con UML, sistema avanzado de reutilización, trabajo colaborativo, soporte a requisitos, gestión de riesgos, técnicas de estimación de proyectos, patrones de diseño, comparador de Modelos UML, Generación de Códigos y esquemas, Gestión de casos de prueba, Sistema de Foros Integrado, Conexión con www.umlmodels.org, y traza completa entre todos los elementos del ciclo de vida; resumiendo swREUSER representa el núcleo de una familia completa de herramientas.

Alta precisión semántica

Uno de los principales objetivos de la herramienta es la representación del metamodelo de UML 1.5 de forma precisa tanto en la parte semántica del estándar como en la parte relativa a la notación. Se ha llevado un cuidado especial para realizar todas las restricciones del estándar y crear así, una interface amigable

No es recomendado el cumplimiento completo de las restricciones y reglas bien formadas del estándar UML debido a la complejidad que supondría a los diseñadores; sin embargo, swREUSER quiere ser la herramienta más semántica del mercado. Para evitar esta paradoja, algunas restricciones de UML se han relajado durante la fase de modelado, haciendo de swREUSER una herramienta flexible y fácil de utilizar. Con todo, se ha añadido un compilador semántico que permite descubrir las desviaciones entre el modelo creado y el estándar UML (incluyendo su metamodelo, restricciones y reglas bien formadas).

Productividad

Sistema avanzado de reutilización

Uno de los factores para incrementar la eficiencia y permitir el ahorro de costes dentro del proceso de desarrollo es la reutilización del software. Además es importante tener en cuenta, que cuanto antes se apliquen estas políticas dentro del ciclo de vida, mayor será el beneficio

swREUSER permite almacenar el contenido de sus modelos de Análisis, Diseño, Requisitos, Riesgos, Pruebas y Estimación.

La potente herramienta de recuperación que incluye en entorno CAKE permite acceder fácilmente a los modelos almacenados. De este modo, el lugar de usar complejos sistemas para acceder a proyectos previos, recordando la ubicación de archivos, su creador… cualquier usuario con los permisos suficientes podrá crear un simple consulta y acceder a aquellos proyectos que incluyan requisitos, riesgos, modelos UML… similares a la consulta. Todo ello se llevará a cabo utilizando la propia herramienta CASE, por lo que no se requiere de nuevos entornos para la recuperación. Así pues, se para de una recuperación add-hoc a una reutilización sistemática de software.

El interface de recuperación y reutilización es sencillo, las herramientas de Procesamiento de Lenguaje Natural (PLN) se utilizan para requisitos y riesgos, mientras que Teoría de Grafos es utilizada para resolver consultas que involucren diagramas UML.

Por último, destacar la facilidad por parte del usuario de llevar la información de la capa conceptual (ontología; dmCAKE y tmCAKE pueden ayudar a esta definición) hasta la capa de software.

Vocabulario controlado

swREUSER permite al usuario definir cuál es el vocabulario válido para el proyecto y así establecer el dominio del mismo. Para ello, es importante tanto dar el nombre correcto a cada uno de los elementos de UML como comprobar si los nombres que ya ha aportado respetan este vocabulario controlado.

Todo este vocabulario proviene de una etapa de modelado de dominios y es un aspecto clave en la metodología de reutilización, ya que hacen que las herramientas de PLN trabajen correctamente, además de permitir el envío de información entre la capa conceptual y la capa de software.

Patrones de diseño

swREUSER incluye una potente herramienta de patrones de diseño, pero con un uso muy sencillo, con la que introduce patrones de GoF dentro de sus modelos. También le permite definir sus propios patrones para compartirlos en su organización

Trabajo colaborativo

swREUSER posee una herramienta de fusión que permite el trabajo colaborativo, ya que permite integrar la información de dos o más contribuyentes.

El software le permite fusionar información sobre riesgos, requisitos, estimación o testeo y a la vez se ha implementado un sistema avanzado para fusionar información procedente del modelado con UML.

También se tiene en cuenta si un elemento ha sido modificado por más de un contribuyente. Para facilitar este trabajo, el módulo es configurable permitiendo indicar si se desea que prevalezcan las decisiones de un determinado contribuyente, o si se desea una resolución manual de cada caso dudoso.

Comparador de modelos UML

swREUSER permite comparar dos modelos UML, obteniendo un porcentaje de similitud entre ambos y una lista con las diferencias que existen entre ambos modelos. Esta comparación se lleva a cabo utilizando el nombre de cada modelo, y no sus identificadores, lo que le permite comparar modelos que no hayan tenido un origen común.

Soporte a Requisitos

Dado que a la hora de llevar a cabo un proyecto software es importante representar los requisitos del sistema de forma precisa y poder referenciar otros elementos y fases de proyecto con dichos requisitos.

swREUSER cumple esta funcionalidad ya que permite definir y gestionar los requisitos del usuario, del sistema u otros. Además el software permite gestionar la siguiente información:

- Información general de los requisitos: fecha de creación, autor, estado...
- Versionado: todas las versiones previas a un requisito están almacenadas.
- Gestión del estado: permitiendo almacenar en qué estado se encuentra el requisito y el quién, cuándo y por qué se ha llevado a cabo cada cambio de estado.
- Requisitos relacionados: permitiendo pasar de un requisito a sus relacionados.
- Ficheros externos relacionados: almacenando enlaces a los documentos que describen más la detalle cada requisito.
- Traza: relación bi-direccional entre un requisito y el resto de elementos del modelo SwREUSER, incluye una vista dual, lista total de requisitos, o vista jerárquica de los mismos. Además, incluyendo los requisitos en la fase de indización y recuperación. Sin coste adicional podrá llevar a cabo actividades sistemáticas de reutilización. Finalmente, gracias a las capacidades avanzadas de procesamiento de textos, el software permite analizar de forma automática un documento textual y extraer requisitos a partir de él.

Fiabilidad

Gestión de riesgos.

SwREUSER incluye un modulo para la Gestión de Riesgos que permite hacer un seguimiento y control de los riesgos que amenazan nuestros proyectos de software.

Por cada riesgo que pueda surgir, la herramienta es capaz de almacenar la siguiente información:

- Información propia del riesgo: nombre, tipo, periodo posible de ocurrencia...
- Información sobre su retirada: responsable y estrategia para evitar el riesgo, actividades llevadas a cabo para remediarlo...
- Análisis post-mortem: antes de concluir el proyecto el jefe de proyecto debe recapacitar sobre una serie de cuestiones para comprobar si todo ha sido abordado de forma correcta. Esta información, junto con las capacidades de recuperación de SwREUSER, tendrá un gran valor para futuros proyectos.
- Factor de prioridad: permite indicar el orden en qué deben tratarse los riesgos a partir de cálculos probabilísticos, de impacto y coste de retirada.
- Riesgos relacionados: permite ligar riesgos entre sí.
- Ficheros externos relacionados: almacenando enlaces a los documentos que describen más al detalle cada riesgo, sus actividades, estrategia de mitigación...
- Traza: bi-direccional con cualquier elemento del proyecto.

Control

Técnicas de estimación de proyectos.

Las técnicas de estimación de proyectos se llevan a cabo para mejorar las técnicas de procesos. SwREUSER dispone de una herramienta que permite estimar la duración de los proyectos y el número de recursos empleados.

Esta herramienta combina dos técnicas de estimación diferentes, la combinación clásica de Puntos de Función y COCOMO II y una técnica orientada a objetos conocida como POP (Predictive Objects Points), en la cual no hay que realizar ningún cálculo, ya que la propia aplicación SwREUSER lo genera a partir de la información de las clases UML diseñadas y sus operaciones. Finalmente, también se han modelado las métricas MOOD y MOOSE en base a la información orientada a objetos incluida en la fase de modelado UML con SwREUSER.

Planificación

Proceso de planificación.

El proceso de planificación es una de las fases más relevantes y con más importancia dentro del ciclo de vida de desarrollo de software. SwREUSER incluye la siguiente funcionalidad a través de las técnicas de estimación y control y de la conexión con MS Project.

SwREUSER con la conexión con MS Project, permite crear tareas dentro de un diagrama Gantt de planificación. Estas tareas pueden proceder de dos etapas diferentes:

Los requisitos de alto nivel, que normalmente corresponden con ciertas tareas programadas en nuestro diagrama Gantt.

Las actividades de mitigación de los riesgos, que deben tenerse en cuenta en la planificación, así como sus costes y recursos involucrados...

Corrección

Gestión de casos de prueba.

Testeo y pruebas es una de las fases más importantes del ciclo de vida del software, ya que cualquier software debe llevar a cabo todas las pruebas necesarias para asegurar la fiabilidad del software.

SwREUSER define tanto las test suites, como las pruebas que estos deben incluir, de esta forma, se refleja el quién, cuándo y cómo se deben desarrollar las pruebas al sistema, evitando así situaciones inesperadas tras la puesta en producción.

La información sobre las anomalías encontradas en el proyecto, junto con la información sobre su seguimiento se puede gestionar y trazar contra el resto de elementos y artefactos del proyecto software.

Compartición del conocimiento

Sistema de foros integrados.

Ya que el conocimiento es poder, es práctico compartir el conocimiento entre los diferentes recursos humanos ya que este aspecto hará aumentar sus beneficios.

De esta manera, SwREUSER está integrada con el sitio Web UML Models: http://www.umlmodels.org.

Esto permite que los usuarios puedan acceder directamente desde la herramienta, a la siguiente funcionalidad y así pueden enviar mensaje al foro Web; crear, acceder y gestionar mensajes; tener soporte a múltiples foros; anexar modelos creados con swREUSE, así como cualquier otro tipo de archivo; acceso rápido los mensajes y a las respuestas de estos; búsqueda por keywords dentro de los mensajes del foro; crear tu propio sistema de foros en un repositorio local en su organización, con acceso exclusivo para sus colegas; organizar los mensajes en una sección personal denominada MyModels y por último recibir alertas cuando sus mensaje son respondidos.

Accesibilidad

http://www.umlmodels.org

http://umlmodels.org le permite compartir su conocimiento de modo globalizado.

Por lo tanto, portal indicado incluye las siguientes funcionalidades:

- Foro: incluye toda la funcionalidad descrita anteriormente.
- Envío de modelos al repositorio: SwREUSER permite enviar sus modelos al repositorio de umlmodels para que puedan ser accedidos por todo el mundo.
- Acceso al repositorio de modelos: permitiendo establecer consultas desde SwREUSER que serán resueltas en el servidor umlmodels y donde las respuestas son modelos de software. Toda esta funcionalidad puede ser accedida tanto desde dentro de la herramienta SwREUSER como directamente desde la Web.

Organización

Vista de la solución.

Un proyecto software además de los requeridos “documentos de software” existen otras fuentes electrónicas como MS Word, MS Excel, PDF, MS Proyect,... estos son de vital importancia para almacenar requisitos de usuarios, estimaciones, planificaciones...

CAKEStudio ofrece una “vista de la solución” que permite organizar todas las fuentes relevantes para su proyecto, así cualquier tipo de archivo puede formar parte de la solución, incluso, cualquier otro modelo creado por SwREUSER también puede formar parte de la solución.

Esta vista también permite discriminar entre los archivos propios de la solución y aquellos que son de utilidad para la solución pero que no ha sido creado dentro de ella. Este otro tipo de archivos se puede fácilmente incluir a la solución como referencia.

De esta forma, utilizando la vista de la solución, se permite abrir un documento existente, cambiar el nombre a un documento, observar las propiedades de un documento, incluir/excluir documentos de la solución, organizar los documentos en base a carpetas, indexar un documento dentro de un repositorio, buscar en el repositorio por documentos similares a uno dado, o similares a una consulta.

Valor añadido

Generación de código y esquemas.

SxREUSER dispone de técnicas avanzadas para la generación de código para extraer la máxima semántica posible del modelo de origen.

Los lenguajes que se han tenido en cuenta son: Java, Visual Basic .net, XML Schemas

Generación de Clases e Interfaces:	Generación de Asociaciones, teniendo en cuenta:
- Atributo	- Multiplicidades
- Operaciones	- Navegabilidad
- Métodos	- Otras propiedades: Changeability…
- …	Generación de esquemas XML
Generación de jerarquías	Ingeniería Inversa de Java

Tabla 2: Generación de código y esquemas

Compatibilidad con otras herramientas

SwREUSER utiliza XMI como medio para almacenar su información. Sin embargo también le permite almacenar sus archivos en formato mdl completamente compatible con Rational Rose.

De la misma manera, SwREUSER permite importar modelos generados con Rational Rose.

Una vez incorporado el modelo de Rose, los procesos de reutilización le permitirán reutilizar modelos creados en Rational Rose como si fuesen modelos propios de SwREUSER.

Plantillas de modelado

En el momento del arranque, SwREUSER le ofrece la posibilidad de comenzar con un modelo vacío o, por el contrario, arrancar con ciertos datos (estereotipos, restricciones, tipos de datos, tipos de requisitos y de riesgos...) ya creados. Esta información se almacena en plantillas.

SwREUSER proporciona diversas plantillas de modelado (UML, Java, .Net...) sin embargo, le permite crear sus propias plantillas para que sean utilizadas una y otra vez.

- Plantilla para UML
- Plantilla para VB .Net
- Plantilla para Java
- Plantilla para C#
- Plantilla para Casos de Abuso
- Plantilla para ODP
- Plantillas personalizadas

Múltiples modelos

SwREUSER le permite abrir varios modelos de software dentro de la misma instancia de la herramienta; de este modo, le será más fácil reutilizar artefactos entre dos modelos.

Sistema avanzado de Copy & Paste

swREUSER le permite copiar elementos de un modelo y pegarlo en el mismo modelo o en otros modelos. Este proceso puede ser rotacional (el número de elementos no se altera, y simplemente hay un elemento más en un diagrama) o semántico (se crean nuevo elementos con el pegado).

Potencial

En próximas versiones.

Generador de documentación: permite crear de forma sencilla documentación en formato MS Word en base a sus plantillas y sus estándares. Por tanto es conveniente, crear plantillas e insertar la información procedente de requisitos, riesgos, pruebas y modelado.

Generación de esquemas relacionales: de forma similar a como se generan esquemas XML, SwREUSER creará esquemas relacionales a partir de la información encerrada en los diagramas de clases. Estos esquemas serán compatibles con las bases de datos más utilizadas del momento: SQLServer, Oracle...

Generación de requisitos a través de un interface Web: permite a los analistas compartir sus requisitos con sus clientes u otros interesados. Estos interesados podrán acceder a los requisitos a través de Web, crear nuevos requisitos, crear nuevas versiones de los requisitos, modificar sus estados...

Gestión de tareas: permite gestionar la planificación del proyecto en base a diagramas Gantt.

Gestión de presupuesto: a partir de la técnica de la Gestión del Valor Conseguido podrá saber si sus proyectos se encuentran dentro de márgenes o, por el contrario, saber cuál será la fecha de finalización y el coste final más probable. Para ello se hace uso de diagramas WBS (Work Breakdown Structure).

2.4 Proyectos similares

El desarrollo de grandes y complejos sistemas requiere el uso de herramientas CASE para la gestión de requisitos software, estos son de gran importancia porque establecen las necesidades del usuario y por consiguiente la comprensión por parte del analista y/o diseñador.

La clave del éxito de cualquier proceso de gestión de requisitos es la trazabilidad, es decir, la técnica que se utiliza para proporcionar relaciones entre requisitos, diseño e implementación de un sistema para poder gestionar cualquier tipo de cambio y asegurar el éxito de los sistemas entregados.

Las herramientas disponibles hoy en día tienen, principalmente, un enfoque sobre la gestión de la información de la gestión de requisitos, es decir, trazabilidad y la organización (con algunas excepciones). Las herramientas varían en el nivel de soporte a estas actividades. La mayoría de las herramientas proporcionan vistas de la gestión de requisitos orientadas al texto (por ejemplo, DOORS, RTM, Document Director, y otras). Pocas herramientas tienen vistas de la gestión de requisitos orientadas a modelos (como por ejemplo, CORE y RDD-100). Otro método (diferente a documentos que describan el diseño o modelos que describan funciones) es capturar el diseño desde diferentes perspectivas y bajar con los requisitos a través de las diferentes alternativas de implementación (SLATE es un ejemplo de este método).

Entre las capacidades organizativas de las herramientas incluye agrupamiento de los requisitos funcionales y asignación de palabras clave y atributos a los requisitos y establecer enlaces para facilitar la búsqueda de conjuntos de requisitos.

Otras características disponibles en herramientas de gestión de requisitos incluye la generación de informes a medida y la proporción de interfaces con otros sistemas y herramientas de la ingeniería del software. También, muchas herramientas proporcionan interfaces de usuario gráficos con múltiples-ventanas

Entre las distintas aplicaciones similares destaca:

Requisite Pro. Herramienta que permite administrar requisitos y consta de las siguientes características principales:

Posibilidad de manejar métricas desde diferentes perspectivas.

Soporte de trazabilidad para estimación de impacto y seguimiento de los cambios.

Requisite Pro es parte de la Solución Rational en lo referente a Administración de Requisitos. Se complementa con Rational Rose (Análisis y Diseño), Rational Soda (Documentación de Modelo) y Rational Suite TestStudio (Herramientas para la Automatización de las pruebas de Software), Rational ClearCase (Herramienta para el control y administración de versiones), y Rational Unified Process (Asistente de Metodología y Procedimientos de Desarrollo de Software). Indudata presta el servicio de consultoría y Entrenamiento para su correcta utilización, así como el servicio de Outsourcing en Desarrollo de Software utilizando esta herramienta.

DOORS.- Es una suite para la Administración de Requisitos. Ofrece tres ambientes para la edición de requisitos:

DOORS, ambiente de formas, orientado a bases de datos que permite a los usuarios capturar, vincular, llevar el rastro, analizar y manejar la información de todos los requisitos de un proyecto, de tal forma que asegure que el proyecto cumpla con los requisitos iniciales y los estándares. Tiene el poder y la escalabilidad para manejar los proyectos grandes, complejos con muchos usuarios concurrentes que trabajan sobre una red. Dirigida a las necesidades específicas de gerentes, directores de proyectos, desarrolladores y usuarios finales a través del ciclo de vida de un proyecto.

DOORSnet permite al equipo de trabajo acceder a los datos de los requisitos, modificarlos, consultar, planear cambios, a cualquier hora y desde cualquier lugar, usando un Browser estándar a través de Internet, o de la Intranet Corporativa o desde una red LAN.

DOORSrequireIT ambiente basado en Microsoft Word para usuarios no técnicos que están acostumbrados a trabajar con Microsoft Word haciendo fácil la participación en un proyecto.

IRQA.- Es una herramienta CASE para Ingeniería de Requisitos especialmente diseñada para soportar las actividades realizadas en el proceso de especificación de sistemas, ayudando a los analistas en las siguientes tareas:

- Captura de requisitos.
- Análisis de requisitos.
- Gestión de requisitos.
- Organización y clasificación de los requisitos.
- Especificación de la solución.
- Gestión de pruebas de aceptación.

3 Tecnología Utilizada

3.1 Herramientas y tecnologías empleadas

3.1.1 Visual Basic .NET

.NET es toda una nueva arquitectura tecnológica, desarrollada por Microsoft para la creación y distribución del software como un servicio. Esto quiere decir, que mediante las herramientas de desarrollo proporcionadas por esta nueva tecnología, el programador puede crear aplicaciones basadas en servicios para la web.

Las características principales que conforman .NET son las siguientes:

- La plataforma .NET Framework, que proporciona la infraestructura para crear aplicaciones y el entorno de ejecución para las mismas.
- Los productos de Microsoft enfocados hacia .NET, entre los que se encuentran Windows .NET Server, como sistema operativo que incluirá de forma nativa la plataforma .NET Framework; Visual Studio .NET, como herramienta integrada para el desarrollo de aplicaciones; Office .NET; b.Central para .NET, etc.
- Servicios para .NET desarrollados por terceros fabricantes, que podrán ser utilizados por otras aplicaciones que se ejecuten en Internet.

Gracias a .NET y a su modelo de desarrollo basado en servicios, se flexibiliza y enriquece el modo en el que hasta ahora se construían aplicaciones para Internet. La idea que subyace bajo esta tecnología , es la de poblar Internet con un extenso número de aplicaciones, que basadas en servicios para la Web (Web Services), formen un marco de intercambio global, gracias a que dichos servicios están fundamentados en los estándares SOAP y XML, para el intercambio de información.

En este sentido, un programador puede crear Web Services para que sean utilizados por sus propias aplicaciones a modo de componentes (pero de una forma mucho más avanzada que empleando el modelo COM clásico), siguiendo una estructura de programación ya conocida.

Elementos principales.

La siguiente tabla proporciona definiciones y ejemplos para cada uno de los principales elementos de la plataforma .NET

Elemento	Definición	Ejemplos
Herramientas de Desarrollo	Interfaces y herramientas de programación para diseñar, crear, ejecutar e implantar soluciones para la plataforma .NET	.NET FrameWork; Visual Studio.NET
Servidores	Infraestructura para generar, implantar y operar soluciones para la plataforma .NET	Microsoft Windows 2003 Server; Servidores Microsoft .NET Enterprise Servers
Servicios Web XML	Conjunto centralizado de servicios predefinidos para realizar tareas habituales y rutinarias y el medio para que los desarrolladores creen sus propios servicios.	Servicios personalizados Microsoft .NET
Clientes	Dispositivos ejecutando sistemas operativos que se integran e interactúan con el resto de elementos de .NET	Microsoft Windows CE para dispositivos de mano; Microsoft Windows XP para PC's
Experiencias de Usuario	Software cliente convencional integrado con servicios Web XML para presentar todo lo que los usuarios necesitan de un modo que tenga sentido para ellos.	Futuras versiones de Microsoft bCentral; Microsoft MSN□

Tabla 3: Elementos de la plataforma .NET

.NET Framework

El .NET Framework es un conjunto de servicios de programación diseñados para simplificar el desarrollo de aplicaciones sobre el entorno distribuido de Internet. El .NET Framework tiene dos componentes principales: el Common Language Runtime y la biblioteca de clases.

Cuando creamos una nueva aplicación Windows en Visual Basic .NET, se nos proporciona un código inicial que incluye el espacio de nombres System.Windows.Forms y la clase Form. Con esta clase, podemos crear fácilmente ventanas, botones, menús, barras de herramientas y otros elementos de pantalla. Cuando compilamos la aplicación, el código se traduce al lenguaje común del entorno de ejecución, Microsoft Intermediate Language (MSIL). Una vez la aplicación se ha compilado, el entorno de ejecución gestiona su ejecución.

El entorno de ejecución incluye una característica denomina compilación just-in-time (JIT), que traduce código MSIL al lenguaje máquina del sistema en el que la aplicación se ejecutará. Cuando un dispositivo cliente con la plataforma .NET lanza la aplicación en Visual Basic .NET, se ejecuta en el lenguaje máquina del sistema cliente y puede integrarse totalmente e interactuar con otras aplicaciones y servicios basados en .NET independientemente del lenguaje en el que hayan sido desarrollados.

Para entender cómo funciona el .NET Framework, debemos estar familiarizados con la siguiente terminología:

- Clase: Una clase en una entidad de programación con nombre que consta de un conjunto común de métodos, propiedades y atributos. Por ejemplo Form es una de las clases del espacio de nombres System.Windows.Form que se utiliza para crear formularios Windows Form.
- Espacio de nombres: identifica una colección de clases relacionadas y/u otros espacios de nombres del .NET Framework. Algunos ejemplos de espacios de nombres incluyen:
    - System.
    - System.Windows.Forms
    - Biblioteca de clases: es una colección completa orientada a objetos de clases reutilizables y organizadas en espacios de nombres jerárquicos en base a su funcionalidad. Podemos utilizar la biblioteca de clases para desarrollar aplicaciones que abarcan desde las aplicaciones cliente tradicional hasta las aplicaciones basadas en las últimas innovaciones proporcionadas por ASP.NET y los servicios Web XML.
    - Common Language Runtime: es la base del .NET Framework. En el entorno .NET, los programadores desarrollan aplicaciones en el lenguaje compatible con .NET que elijan; el código se compila en MSIL, y el entorno de gestión gestiona y ejecuta el código compilado.

¿Qué es Visual Studio .NET?

Visual Studio .NET es un entorno de desarrollo integrado que nos ayuda a diseñar, desarrollar, depurar e implantar con rapidez soluciones basadas en el .NET Framework. Podemos acceder a un conjunto común de herramientas, diseñadores y editores desde cualquiera de los lenguajes de programación de Visual Studio .NET. Podemos crear aplicaciones Windows Forms y Web Forms que integren datos y lógica de negocio.

3.1.2 XML (extensible markup language)

XML, siglas en inglés de Extensible Markup Language (lenguajes de marcas extensibles), es un metalenguaje extensible desarrollado por el Word Wide Web Consortium (W3C). Es una simplificación y adaptación de SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido solo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto , hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que se complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Las ventajas que presenta XML son las siguientes:

- Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan bugs y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

Así pues, la tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada implica que sus partes estén bien definidas donde esas partes se componen a su vez de otras partes. Entonces se obtendrá un árbol estructurado de información, a partir de elementos y etiquetas. Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Una parte de información con un sentido claro y definido. Las etiquetas tiene la forma <nombre> , donde nombre es el nombre del elemento que se está señalando.

Un documento XML bien formado son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (parser) que cumpla con la norma. De este modo, los documentos han de seguir una estructura estrictamente jerárquica con la que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.

3.1.3 Microsoft Windows XP Profesional

El desarrollo de todo el proyecto ha sido llevado a cabo bajo este sistema operativo. Este sistema operativo proporciona un nuevo estándar de fiabilidad y rendimiento. Windows XP profesional integra los puntos fuertes de Windows 2000 Profesional, como la seguridad basada en estándares, la capacidad de administración y la fiabilidad, con las mejores características comerciales de Windows 98 y Windows Me (por ejemplo, Plug and Play, una interfaz de usuario más sencilla y novedosos servicios de soporte). Este nuevo sistema operativo aumenta la capacidad informática al tiempo que reduce el coste total de los equipos.

3.1.4 Microsoft Project

Es un software de administración de proyectos diseñado, desarrollado y comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

Este software de gestión ha sido empleado para llevar a cabo la planificación del proyecto, así como la asignación de tiempos y recursos del mismo.

3.1.5 Microsoft Office

Para la realización de la documentación, elaboración de gráficos, presupuesto y presentación se han empleado algunas de las utilidades dentro de la suite ofimática Microsoft Office tales como Microsoft Word, Microsoft PowerPoint y Microsoft Excel.

4 Objetivos

4.1 Especificación de casos de uso

Un caso de uso es una descripción de una secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad. Un caso de uso modela la interacción entre el usuario y sistema, pero para comprender mejor lo que significa esto, vamos a definir lo que es un escenario.

Un escenario, es una secuencia de pasos que describen la interacción entre el usuario y el sistema. Por lo tanto, cada escenario es algo que puede ocurrir en el sistema con el uso de éste. De modo que, un caso de uso es un conjunto de escenarios que pueden ser agrupados para la consecución de un fin común. Normalmente uno de estos escenarios representa el caso más típico del uso de una parte determinada del sistema y los demás escenarios correspondientes al mismo caso de uso, representan cursos raros o posibilidades de realizar la misma operación utilizando para ello caminos alternativos.

Para representar de forma gráfica los casos de uso, se utiliza el lenguaje de modelado UML.

En un diagrama de casos de uso, además de aparecer representados los casos de uso, aparecen representados también los actores. Un actor representa un rol jugado por una persona o cosa que interactúa con el sistema. Una misma persona o cosa puede desempeñar varios roles. Los actores son externos al sistema.

En el siguiente diagrama de casos de uso, se recoge la funcionalidad a la que puede acceder cada uno de los distintos tipos de usuarios identificados para el sistema. En diagramas posteriores dentro de este mismo punto se detallarán los casos de uso en los que se descomponen algunos de los que podemos ver en este diagrama.

En primer lugar, antes de explicar el diagrama expuesto y entrar al detalle a partir de las especificaciones siguientes, es importante destacar que este diagrama de casos de uso muestra cada una de las funcionalidades del sistema para gestionar proyectos, es decir el proyecto en cuestión será tratado por el sistema para ser gestionado a partir de las distintas funcionalidades ofrecidas por la aplicación. A continuación, podemos ver una figura para aclarar en la situación en la que se encuentra dicho diagrama de elementos creado y en qué punto nos encontramos:

Ilustración 8: Diagrama de Casos de Uso 1

Caso de Uso Gestionar Plantilla

El caso de uso Gestionar Plantilla hace referencia a la funcionalidad que el sistema ofrece al Analista para interactuar con las plantillas generadas, inicialmente verificadas por el Jefe de proyecto. De este modo el Analista podrá insertar, modificar y eliminar los datos de dichas plantillas.

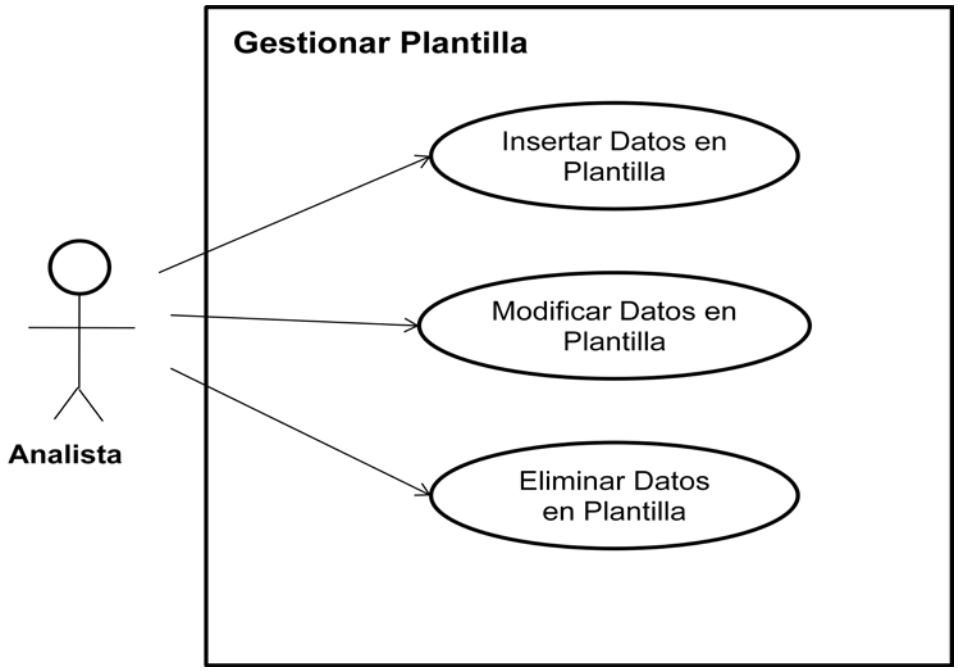


Ilustración 9: Diagrama de Casos de Uso 2

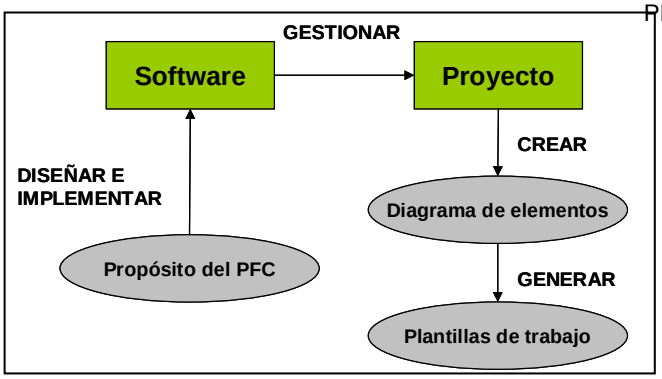


Ilustración 10: Propósito del proyecto

El diagrama de casos de usos expuesto, recoge las distintas funcionalidades del software a llevar a cabo, en el cual, solo se hace partícipe a dos tipos de actores, que toman un role determinado (analista/jefe de proyecto), lo cual le permite realizar distintas funcionalidades.

De este modo, el Jefe de proyecto se encarga en primer lugar de pensar y diseñar el diagrama de elementos y generar las plantillas en función del diagrama de elementos creado, de este modo sólo verifica la exactitud de la vista de dichas plantillas conforme a la idea esperada y la misión del Analista de software se basa en la gestión de las plantillas generadas por el sistema, es decir, la gestión de la información contenida en las plantillas, pudiendo insertar, modificar o eliminar datos de las plantillas seleccionadas. Además, el Analista puede guardar dichas plantillas para consultar en futuras ocasiones bien para modificar sus datos o para reutilizar su contenido a otros proyectos.

A continuación, para más detalle, se tratarán por separado cada una de las funcionalidades expuestas en el diagrama de casos de uso, indicando la descripción general, los actores que participan para que se pueda llevar a cabo y otros tipos de detalles que pueden resultar de gran utilidad para el entendimiento del proyecto.

4.1.1 Descripción textual de los casos de uso

A continuación se realizará una descripción textual de cada uno de los casos de uso, usando las plantillas que se mostraban en el punto 2.2 de la memoria.

Nombre:	Buscar Diagrama de Clases
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Encontrar un archivo con extensión ".csem" que contenga un diagrama de clases previamente dibujado.	
Actores:	
Jefe de Proyecto / Analista	
Precondiciones:	
El archivo .csem debe estar previamente guardado.	
Flujo Normal:	
- Seleccionar botón para abrir el explorador de Windows. - Seleccionar un archivo .csem y hacer doble clic o clic en el botón abrir.	
Flujo Alternativo:	
No hay	
Poscondiciones:	
Archivo .csem con el diagrama de clases encontrado.	

Tabla 4: Caso de uso. Buscar diagrama de clases

Nombre:	Generar Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Transformación del modelo que recoge el diagrama de elementos previamente creado en plantillas. Cada elemento tendrá una plantilla asociada que incluye información de sus atributos.	
Actores:	
Jefe de Proyecto	
Precondiciones:	
Archivo .csem con diagrama de elementos creado.	
Flujo Normal:	
- Seleccionar la opción de Generar Plantillas sin opción a guardar. - Buscar Archivo .csem - Procesar dicho archivo. - Transformación de dicho archivo a plantillas.	
Flujo Alternativo:	
El archivo .csem está vacío ya que por el momento el Jefe de Proyecto no ha diseñado el diagrama de elementos.	
Poscondiciones:	
Plantillas creadas a partir del diagrama de elementos recogido en el archivo buscado.	

Tabla 5: Caso de uso. Generar Plantilla

Nombre:	Visualizar Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Visualizar cada uno de los elementos del diagrama de elementos en plantillas separadas, cada plantilla mostrará información sobre sus atributos e items si tuviera.	
Actores:	
Jefe de Proyecto / Analista	
Precondiciones:	
Diagrama de elementos creado por el Jefe de Proyecto.	
Flujo Normal:	
El usuario selecciona una plantilla a partir de los botones siguiente y anterior. Se visualiza los datos mostrados en la plantilla en cuestión	
Flujo Alternativo:	
El archivo .csem está vacío, o bien las plantillas no contienen datos insertados.	
Poscondiciones:	
No hay	

Tabla 6: Caso de Uso. Visualizar Plantilla

Nombre:	Generar Plantilla con opción a Guardar
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Generar plantillas de trabajo como el caso de uso tratado anteriormente, pero dando la opción de guardar dichas plantillas.	
Actores:	
Analista	
Precondiciones:	
Archivo .csem con diagrama de elementos creado y plantillas previamente revisadas por el jefe de proyecto.	
Flujo Normal:	
Seleccionar la opción de Generar Plantillas con opción a guardar. Buscar Archivo .csem. Procesar dicho archivo. Transformación de dicho archivo a plantillas con opción a ser guardadas.	
Flujo Alternativo:	
El archivo .csem está vacío ya que por el momento el Jefe de Proyecto no ha diseñado el diagrama de elementos.	
Poscondiciones:	
Plantillas creadas a partir del diagrama de elementos recogido en el archivo buscado.	

Tabla 7: Caso de Uso. Generar Plantilla con opción a Guardar

Nombre:	Guardar Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Guardar las plantillas generadas bajo un mismo archivo, el analista decidirá la ubicación de dicho archivo.	
Actores:	
Analista	
Precondiciones:	
Plantillas previamente generadas con la opción que permite guardar dichas plantillas.	
Flujo Normal:	
- Seleccionar el botón Guardar. - Elegir la ubicación para guardar el archivo con contiene la información.	
Flujo Alternativo:	
No hay	
Poscondiciones:	
Archivo con la información de las plantillas guardado.	

Tabla 8: Caso de Uso. Guardar Plantilla

Nombre:	Insertar datos en Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
El usuario podrá insertar datos en cada una de las plantillas.	
Actores:	
Analista	
Precondiciones:	
Plantillas previamente creadas	
Flujo Normal:	
- Seleccionar la plantilla escogida para insertar los datos con los botones anterior y siguiente. - Insertar los datos para cada una de los campos que compone el nuevo registro. - Registro insertado en la plantilla indicada.	
Flujo Alternativo:	
Sin la existencia de plantillas no es posible insertar registros.	
Poscondiciones:	
Plantilla seleccionada con un nuevo registro insertado.	

Tabla 9: Caso de uso. Insertar datos en Plantilla

Nombre:	Modificar datos en Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
El usuario podrá modificar los datos previamente insertados en las plantillas	
Actores:	
Analista	
Precondiciones:	
Plantillas previamente creadas y con datos insertados.	
Flujo Normal:	
- Seleccionar la plantilla a modificar. - Seleccionar el campo del registro a modificar. - Insertar el nuevo dato en dicho campo.	
Flujo Alternativo:	
Sin la existencia de plantillas y datos insertados no es posible modificar la información.	
Poscondiciones:	
Campo del registro modificado para la plantilla seleccionada.	

Tabla 10: Caso de uso. Modificar datos en Plantilla

Nombre:	Eliminar datos en Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
El usuario podrá eliminar los datos previamente insertados en las plantillas	
Actores:	
Analista	
Precondiciones:	
Plantillas previamente creadas y con datos insertados.	
Flujo Normal:	
- Seleccionar la plantilla donde eliminar datos. - Seleccionar el campo del registro a eliminar. - Eliminar el dato en dicho campo.	
Flujo Alternativo:	
Sin la existencia de plantillas y datos insertados no es posible eliminar la información.	
Poscondiciones:	
Campo del registro eliminado para la plantilla seleccionada.	

Tabla 11: Caso de uso. Eliminar datos en Plantilla

Nombre:	Buscar Plantillas Guardadas
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Encontrar un archivo con extensión ".txt" que contenga la información recogida previamente en las plantillas.	
Actores:	
Jefe de Proyecto / Analista	
Precondiciones:	
El archivo .txt debe estar previamente guardado.	
Flujo Normal:	
- Seleccionar botón para abrir el explorador de Windows. - Seleccionar un archivo .txt y hacer doble clic o clic en el botón abrir.	
Flujo Alternativo:	
No hay	
Poscondiciones:	
Archivo .txt con la información recogida en las plantillas.	

Tabla 12: Caso de Uso. Buscar Plantillas guardadas

Nombre:	Recuperar Plantilla
Autor:	María Marchena Ronda
Fecha:	09/07/2009
Descripción:	
Recuperar y visualizar las plantillas previamente guardadas bajo un archivo con formato xml.	
Actores:	
Analista	
Precondiciones:	
El archivo con formato xml debe estar previamente guardado.	
Flujo Normal:	
- Seleccionar botón para Gestionar Plantillas. - Seleccionar un archivo con formato xml y hacer doble clic ó clic en el botón abrir. - Plantillas recuperadas.	
Flujo Alternativo:	
No hay	
Poscondiciones:	
Plantillas mostradas en base a la información contenida en el diagrama de elementos.	

Tabla 13: Caso de uso. Recuperar Plantilla

4.2 Requisitos de Usuario

La ingeniería de requisitos es la rama de la ingeniería del software que se ocupa de la comprensión y formalización de las necesidades de un proyecto en la primera etapa de desarrollo software.

Surgió como respuesta a la continua preocupación por el fracaso en el correcto desarrollo de programas software. El número de proyectos cancelados era aproximadamente el doble que el de proyectos finalizados con éxito. Esta falta de eficiencia motivó la aparición de la Ingeniería del Software, y dentro de esta la Ingeniería de Requisitos, para mejorar la calidad de los productos software.

La Ingeniería de Requisitos cada vez tiene más importancia debido que es una parte indispensable dentro de un proyecto software, llegando incluso a no terminar un proyecto con éxito si los requisitos están mal especificados y definidos.

Por este motivo, en este apartado se llevará a cabo el proceso de comprensión y formalización de las necesidades que deben satisfacer el sistema; es decir, definiremos qué debe hacer la aplicación a realizar.

4.2.1 Requisitos Funcionales

Los requisitos funcionales marcan el "qué" del sistema, es decir, la funcionalidad del mismo, sin adentrarse en temas de diseño. Estos requisitos son básicamente de información y de operación.

Identificador: SF – 01	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante el comienzo del sistema
Nombre:	Objetivo
Descripción:	Crear una aplicación capaz de gestionar requisitos de un proyecto software a partir de plantillas.

Tabla 14: Requisito SF - 01

Identificador: SF – 02	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante el comienzo del sistema
Nombre:	Dibujar diagrama de elementos
Descripción:	El sistema permite al usuario la creación del diagrama de elementos (diagrama de clases) que define los elementos importantes del proyecto.

Tabla 15: Requisito SF - 02

Identificador: SF - 03	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante el comienzo del sistema, una vez creado el diagrama de elementos.
Nombre:	Crear plantillas a partir del diagrama de elementos
Descripción:	Se crearán plantillas de trabajo, una vez creado el diagrama de elementos. Estas plantillas recogerán todos los atributos de cada uno de los elementos y las distintas relaciones que se establezcan. Este paso será automático.

Tabla 16: Requisito SF - 03

Identificador: SF - 04	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez creadas las plantillas.
Nombre:	Insertar datos en plantillas
Descripción:	El usuario podrá insertar datos en las plantillas creadas, creado un nuevo item en la platilla seleccionada.

Tabla 17: Requisito SF - 04

Identificador: SF - 05	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez creadas las plantillas.
Nombre:	Modificar datos en plantillas
Descripción:	El usuario podrá modificar items ya insertados en las plantillas creadas.

Tabla 18: Requisito SF - 05

Identificador: SF - 06	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez creadas las plantillas.
Nombre:	Eliminar datos en plantillas
Descripción:	El usuario podrá eliminar items ya insertados en las plantillas creadas.

Tabla 19: Requisito SF - 06

Identificador: SF - 07	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez creadas las plantillas e introducidos los datos.
Nombre:	Trazar relaciones entre ítems de las plantillas.
Descripción:	El usuario podrá establecer relaciones entre los datos introducidos en las plantillas, de este modo podrá relacionar información que le será de gran utilidad.

Tabla 20: Requisito SF - 07

Identificador: SF - 08	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez que se haya insertado datos en las plantillas y los mismos se encuentren relacionados.
Nombre:	Consultar relaciones entre ítems.
Descripción:	El usuario podrá visualizar a través del interfaz las distintas relaciones entre los datos insertados.

Tabla 21: Requisito SF - 08

Identificador: SF - 09	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema, una vez creadas las plantillas con sus ítems insertados y las relaciones establecidas.
Nombre:	Visualizar datos sobre las plantillas.
Descripción:	El usuario podrá visualizar los datos ya insertados en las plantillas creadas y navegar a través de las distintas plantillas entre los ítems trazados.

Tabla 22: Requisito SF – 09

Identificador: SF - 10	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Mensajes del sistema.
Descripción:	Todas las operaciones en el sistema generan algún tipo de mensaje para informar al usuario si la operación se ha realizado con éxito o si se ha producido un error.

Tabla 23: Requisito SF - 10

Identificador: SF - 11	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Guardar plantillas
Descripción:	Las plantillas generadas por el sistema puede ser guardada por el usuario en la ubicación que el desee.

Tabla 24: Requisito SF - 11

Identificador: SF - 12	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Recuperar plantillas
Descripción:	El usuario puede recuperar las plantillas guardadas.

Tabla 25: Requisito SF - 12

Identificador: SF - 13	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Modificar datos sobre plantillas recuperadas
Descripción:	El usuario tiene la opción de modificar datos de las plantillas utilizadas y guardadas con anterioridad.

Tabla 26: Requisito SF - 13

Identificador: SF - 14	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Comtemplar la relación de herencia en plantillas.
Descripción:	Las plantillas mostrarán las posibles relaciones de herencia que contenga el diagrama de elementos en los atributos de las clases.

Tabla 27: Requisito SF - 14

4.2.2 Requisitos de Restricción

Los requisitos no funcionales restringen el "cómo" del sistema, es decir, detallan un posible prediseño del sistema en cuestión.

Identificador: SNF - 01	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la creación del diagrama de elementos.
Nombre:	Restringir la elección de elementos (clases) del diagrama de elementos.
Descripción:	El usuario tiene restringido el uso de elementos a la hora de establecer su diagrama de elementos, estas son: Requisito, Caso de Uso, Prueba, Riesgo y Gestión de Configuración (versión).

Tabla 28: Requisito SNF - 01

Identificador: SNF - 02	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la creación del diagrama de elementos.
Nombre:	Restringir el uso de relaciones en el diagrama de elementos.
Descripción:	El usuario tiene restringido el uso de relaciones a la hora de establecer su diagrama de elementos, estas son: asociación y herencia.

Tabla 29: Requisito SNF - 02

Identificador: SNF - 03	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema. Antes del diagrama de elementos y del uso de las plantillas.
Nombre:	Perfiles de usuario.
Descripción:	Existirán dos tipos de perfil en el sistema:
	Jefe de proyecto (administrador)
	Analista-Programador (usuario)

Tabla 30: Requisito SNF - 03

Identificador: SNF - 04	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Menú de acceso.
Descripción:	La aplicación a desarrollar deberá ofrecer en todo momento el menú de opciones que promueva su usabilidad. Este menú, será necesario sobre todo para tratar las plantillas y el fácil acceso a las distintas opciones facilitará el uso de las mismas.

Tabla 31: Requisito SNF - 04

Identificador: SNF - 05	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante el análisis, diseño e implementación del sistema.
Nombre:	Entorno de codificación.
Descripción:	El sistema a desarrollar será codificado empleando como lenguaje de programación VB.net

Tabla 32: Requisito SNF - 05

Identificador: SNF - 06	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante el análisis, diseño e implementación del sistema.
Nombre:	Entorno de instalación.
Descripción:	El sistema creado formará parte del software implantado SwReuser, siendo un módulo más del software.

Tabla 33: Requisito SNF - 06

Identificador: SNF - 07	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Tiempo de respuesta a una operación de un usuario.
Descripción:	El sistema da respuesta a cualquiera de las operaciones que realice el usuario en un tiempo no superior a 10 segundos.

Tabla 34: Requisito SNF - 07

Identificador: SNF - 08	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Memoria
Descripción:	El equipo debe disponer de unos niveles óptimos de memoria.

Tabla 35. Requisito SNF - 08

Identificador: SNF - 09	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Capacidad de procesamiento.
Descripción:	El equipo debe disponer de un buen procesador.

Tabla 36: Requisito SNF - 09

Identificador: SNF - 10	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante la ejecución del sistema.
Nombre:	Interfaz amigable
Descripción:	El interfaz es claro, sencillo y tiene una baja curva de aprendizaje. Estas características de calidad junto con sus métricas están definidas en el Plan de aseguramiento de la calidad.

Tabla 37: Requisito SNF - 10

Identificador: SNF - 11	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Cliente ☐ Desarrollador.
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema.
Nombre:	Modificaciones de funcionalidad.
Descripción:	El diseño del sistema se realizará de forma modularizada obteniendo un bajo acoplamiento y una alta cohesión, de tal forma que facilite el mantenimiento del sistema.

Tabla 38: Requisito SNF - 11

4.3 Riesgos del Proyecto

En este apartado se expondrán el conjunto de riesgos a los que ha estado expuesto el desarrollo del sistema; así como aquellos riesgos que se han llegado a materializar. Además de identificar los riesgos, también se definirá un plan de contingencia asociado a cada uno de ellos y las medidas preventivas que se tomaron que no se llegasen a producir.

RPS - 001			
PRIORIDAD	Media	**IMPORTANCIA**	Media
PLAN DE CONTINGENCIA A APLICAR	Retrasar el plazo de entrega del sistema		
DESCRIPCIÓN	Riesgo de no poder cumplir los plazos de entrega fijados		
ACTIVIDAD DE PREVENCIÓN			
Mantener un ritmo constante en el desarrollo del sistema y de la documentación con la capacidad de carga que se ha estimado			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
Partes que no han sido desarrolladas todavía			

Tabla 39: Riesgo RPS - 001

RPS - 002			
PRIORIDAD	Baja	**IMPORTANCIA**	Alta
PLAN DE CONTINGENCIA A APLICAR	Buscar un nuevo tutor dentro del departamento para el seguimiento del proyecto		
DESCRIPCIÓN	Riesgo de que el tutor del proyecto abandone el departamento o la universidad		
ACTIVIDAD DE PREVENCIÓN			
No estimable			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
No aplicable			

Tabla 40. Riesgo RPS - 002

RPS - 003			
PRIORIDAD	Baja	**IMPORTANCIA**	Media
PLAN DE CONTINGENCIA A APLICAR	Emplear los datos que se encuentran en las copias de seguridad		
DESCRIPCIÓN	Riesgo de que se pierda el trabajo debido a problemas externos		
ACTIVIDAD DE PREVENCIÓN			
Respaldo a través de copias de seguridad realizadas con una frecuencia alta.			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
Solo aquellas partes que no han sido guardadas en copias de seguridad se verán afectadas			

Tabla 41: Riesgo RPS - 003

RPS - 004			
PRIORIDAD	Baja	**IMPORTANCIA**	Media
PLAN DE CONTINGENCIA A APLICAR	Se modificará la planificación y estimación de recursos; repercutiendo este hecho en la fecha de entrega del proyecto		
DESCRIPCIÓN	Riesgo de que se modifiquen los objetivos del sistema		
ACTIVIDAD DE PREVENCIÓN			
No aplicable			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
Todas aquellas partes que no han sido desarrolladas por el momento			

Tabla 42: Riesgo RPS - 004

RPS - 005			
PRIORIDAD	Baja	**IMPORTANCIA**	Alta
PLAN DE CONTINGENCIA A APLICAR	Se modificará la planificación y estimación de recursos; repercutiendo este hecho en la fecha de entrega del proyecto		
DESCRIPCIÓN	Riesgo de no disponer de un 100% de los recursos humanos con los que se cuenta para desarrollar el sistema		
ACTIVIDAD DE PREVENCIÓN			
No aplicable			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
Todas aquellas partes que no han sido desarrolladas por el momento			

Tabla 43: Riesgo RPS - 005

RPS – 006			
PRIORIDAD	Baja	**IMPORTANCIA**	Alta
PLAN DE CONTINGENCIA A APLICAR	Se modificará la planificación, dedicando mayor esfuerzo a aquellas partes que tiene que ver con la tecnología en cuestión.		
DESCRIPCIÓN	Riesgo de desconocimiento de la tecnología empleada para el desarrollo del sistema		
ACTIVIDAD DE PREVENCIÓN			
Formación para los desarrolladores que utilicen la tecnología en cuestión			
PARTES DEL PROYECTO A LAS QUE AFECTA EL RIESGO			
Todas aquellas partes que no han sido desarrolladas por el momento y que tiene que ver con la tecnología en cuestión			

Tabla 44: Riesgo RPS - 006

5 Planificación y Presupuesto

Debido a los cambios producidos a lo largo del desarrollo del sistema, se ha tenido que realizar una modificación de la planificación inicial y del presupuesto inicial. Estos cambios se han producido principalmente ante la idea de incluir nuevas funcionalidades al sistema, tales como, guardar las plantillas generadas en una ruta cualquiera y contemplar sobre dichas plantillas, las relaciones de herencia entre las distintas clases del diagrama de elementos.

Ambas funcionalidades han provocado una nueva idea en la planificación y por supuesto, nuevos esquemas para afrontar el presupuesto.

A continuación, en el siguiente punto se detalla la planificación inicial y el presupuesto inicial que se obtuvo y la planificación final y los costes totales que ha supuesto el desarrollo del sistema.

5.1 Presupuesto y Planificación Iníciales

En este apartado se exponen en detalle los costes que sirvieron de base para el cálculo del presupuesto inicial y la planificación inicial que se propuso.

5.1.1 Presupuesto Inicial

Tarifas del personal

El coste del personal por hora de trabajo es el siguiente:

PERFIL	COSTE POR HORA
JEFE DE PROYECTO	15,11 €
DISEÑADOR GRÁFICO	15,11 €
ANALISTA	15,11 €
PROGRAMADOR	15,11 €
RESPONSABLE DE PRUEBAS	15,11 €

Tabla 45: Resumen de las tarifas del personal

Para el cálculo del coste por hora del personal se han tenido en cuenta las siguientes variables:

- Salario neto por persona: 1.150€ (igual para todos los perfiles)
- IRPF (20% neto): 220€
- S. Social 40% (neto + IRPF): 530€
- Salario bruto por persona y mes: 1.900€
- Hora / mes: 160 horas
- Meses laborables: 11 meses
- Horas / año: 1.760 horas
- Salario bruto personal / año: 26.600 (14 pagas)
- Coste bruto hora / persona = (Bruto persona / año) / (Horas / año) = 26600 /1760 = 15,11€.

Tiempo empleado en el proyecto

En la siguiente tabla se desglosan las horas imputadas a cada perfil para cada actividad que se llevará a cabo en el proyecto.

Hemos supuesto que la jornada laboral es de 4 horas.

ACTIVIDAD	JP	DIS	ANA	PROG	R.PRU	TOTAL
ANÁLISIS DEL SISTEMA	80	20	80	0	0	180
DISEÑO DEL SISTEMA	60	80	30	0	0	170
IMPLEMENTACION DEL SISTEMA	0	0	40	380	0	420
PRUEBAS	0	0	0	32	288	320
IMPLANTACION DEL SISTEMA	0	0	30	30	0	60

Tabla 46: Resumen del tiempo empleado en el proyecto

ABREVIATURA	PERFIL
JP	JEFE DE PROYECTO
DIS	DISEÑADOR SOFTWARE
ANA	ANALISTA SOFTWARE
PROG	PROGRAMADOR
R.PRU	RESPONSABLE DE PRUEBAS

Tabla 47: Perfiles involucrados en el proyecto

Gastos de personal imputables al proyecto

De manera similar al apartado anterior, en la siguiente tabla figuran los costes por perfil y actividad que se cargarán al proyecto. Todas las cantidades vienen expresadas en euros.

ACTIVIDAD	JP	DIS	ANA	PROG	R.PRU	TOTAL
ANÁLISIS DEL SISTEMA	1208,8	302,2	1208,8	0	0	2719,8
DISEÑO DEL SISTEMA	906,6	0	453,3	0	0	1359,9
IMPLEMENTACION DEL SISTEMA	0	0	604,4	5741,8	0	6346,2
PRUEBAS	0	0	0	483,52	4351,68	4835,2
IMPLANTACION DEL SISTEMA	0	0	453,3	453,3	0	906,6

Tabla 48: Gastos del personal del proyecto

ABREVIATURA	PERFIL
JP	JEFE DE PROYECTO
DIS	DISEÑADOR SOFTWARE
ANA	ANALISTA SOFTWARE
PROG	PROGRAMADOR
R.PRU	RESPONSABLE DE PRUEBAS

Tabla 49: Perfiles involucrado en el proyecto

Equipos y licencias software

DESCRIPCIÓN		COSTE (EUROS)
EQUIPOS		500,00 €
LICENCIAS SOFTWARE		250,00 €
	TOTAL	750,00 €

Tabla 50: Costes vinculados a equipos y licencias

Material fungible y mantenimiento del local

DESCRIPCIÓN		COSTE (EUROS)
MATERIAL IMPRESO (PAPEL, TÓNER, ENCUADERNACIÓN,ETC.)		150,00 €
MATERIAL DE OFICINA		60,00 €
MANTENIMIENTO LOCAL		40,00 €
	TOTAL	250,00 €

Tabla 51: Costes vinculados a material fungible y mantenimiento del local

Resumen de costes del proyecto

DESCRIPCIÓN	COSTE (EUROS)
GASTOS DEL PERSONAL IMPUTABLES AL PROYECTO	16.167,70 €
EQUIPOS Y LICENCIAS SOFTWARE	750,00 €
MATERIAL FUNGIBLE Y GASTOS DEL LOCAL	250,00 €
OTROS GASTOS	0,00 €
TOTAL	17.167,70 €

Tabla 52: Resumen de costes del proyecto

Resumen del presupuesto

El presupuesto final para el proyecto es el siguiente:

DESCRIPCIÓN	COSTE (EUROS)	
GASTOS DEL PERSONAL IMPUTABLES AL PROYECTO	16.167,70 €	
EQUIPOS Y LICENCIAS SOFTWARE	750,00 €	
MATERIAL FUNGIBLE	250,00 €	
SUBTOTAL	17.167,70 €	
COSTES INDIRECTOS	2.575,16 €	(15% SUBTOTAL)
RIESGO	3.233,54 €	(20% GASTOS PERSONAL)
BENEFICIO	1.742,61 €	(30% (COSTES INDIRECTOS + RIESGO))
TOTAL	24.719,00 €	

Tabla 53: Resumen de presupuesto

Las fórmulas empleadas para el cálculo del presupuesto han sido las siguientes:

- Costes indirectos: se han calculado como el 15% de los costes directos imputados al proyecto.
- Riesgo: para calcular este importe, se ha aplicado un factor del 20% sobre los Gastos del personal, ya que es en este aspecto donde pueden sufrirse mayor número de variaciones.
- Beneficios: los beneficios se han calculado como el 30% de la suma de costes (directos e indirectos) y riesgo.

5.1.2 Planificación Inicial

En el siguiente gráfico se muestra el diagrama de Gantt correspondiente a la planificación inicial que se realizó para el desarrollo del sistema. Como puede observarse, la fecha de comienzo del mismo se remonta a enero del 2009 y su finalización se planificó para julio del mismo año. Como se detallo anteriormente, la demora en la finalización del proyecto se debe a la ampliación de funcionalidades y aspectos del sistema que fueron estudiados a última hora y que se consideraron aconsejables e importantes para el funcionamiento de este.

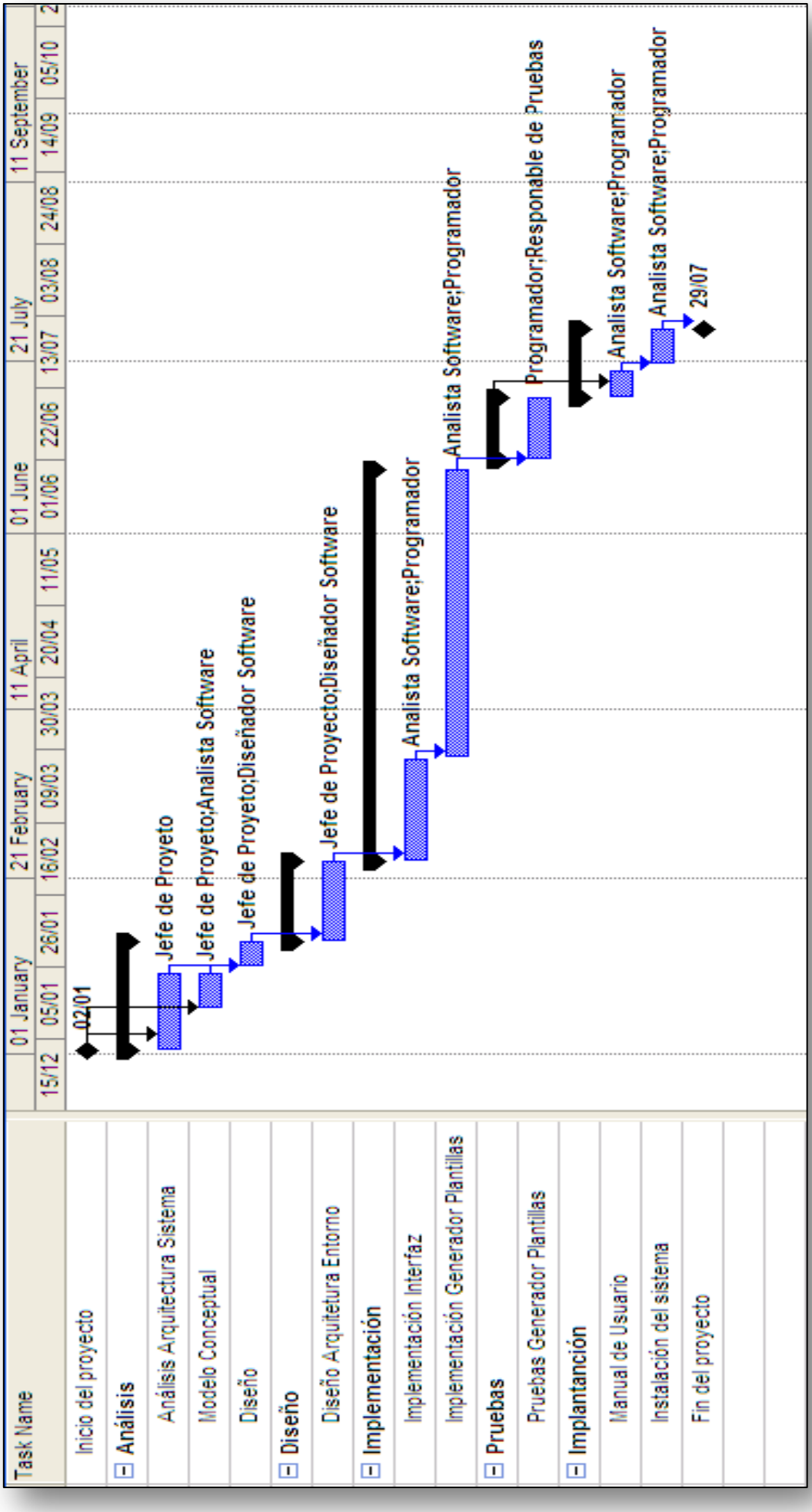


Ilustración 11: Diagrama Gantt. Planificación Inicial

5.2 Presupuesto y Planificación Finales

En este apartado se expondrán en detalle los costes que han servido de base para obtener el coste final del proyecto; así como la planificación que se ha seguido para llevar a cabo el desarrollo del sistema.

5.2.1 Presupuesto Final

En este apartado se expondrán todos los costes que han servido de base para el cálculo del presupuesto final.

Tarifas del personal

El coste del personal por hora de trabajo es el siguiente:

PERFIL	COSTE POR HORA
JEFE DE PROYECTO	15,11 €
DISEÑADOR GRÁFICO	15,11 €
ANALISTA	15,11 €
PROGRAMADOR	15,11 €
RESPONSABLE DE PRUEBAS	15,11 €

Tabla 54: Gastos de personal del proyecto final

Para el cálculo del coste por hora del personal se han tenido en cuenta las siguientes variables:

- Salario neto por persona: 1.150€ (igual para todos los perfiles)
- IRPF (20% neto): 220€
- S. Social 40% (neto + IRPF): 530€
- Salario bruto por persona y mes: 1.900€
- Hora / mes: 160 horas
- Meses laborables: 11 meses
- Horas / año: 1.760 horas
- Salario bruto personal / año: 26.600 (14 pagas)

Coste bruto hora / persona = (Bruto persona / año) / (Horas / año) = 26600 /1760 = 15,11€.

Tiempo empleado en el proyecto

En la siguiente tabla se desglosan las horas imputadas a cada perfil para cada actividad que se llevará a cabo en el proyecto.

Hemos supuesto que la jornada laboral es de 4 horas.

ACTIVIDAD	JP	DIS	ANA	PROG	R.PRU	TOTAL
ANÁLISIS DEL SISTEMA	80	20	85	0	0	185
DISEÑO DEL SISTEMA	60	80	40	0	0	180
IMPLEMENTACION DEL SISTEMA	0	0	40	400	0	440
PRUEBAS	0	0	0	32	300	332
IMPLANTACION DEL SISTEMA	0	0	30	30	0	60

Tabla 55: Gastos de personal de proyecto final

ABREVIATURA	PERFIL
JP	JEFE DE PROYECTO
DIS	DISEÑADOR SOFTWARE
ANA	ANALISTA SOFTWARE
PROG	PROGRAMADOR
R.PRU	RESPONSABLE DE PRUEBAS

Tabla 56: Perfiles involucrados en el proyecto

Según podemos observar, en algunas de las tareas, para algunos de los perfiles, se ha incrementado el número de horas para alcanzar todos los objetivos marcados, tanto los iníciales como los finales que se han considerado tan importantes para la funcionalidad del sistema y que finalmente fueron incluidos como requisito que debe cumplir la aplicación a diseñar.

Cabe destacar también, el hecho de cada uno de los perfiles no ha participado en todas las actividades y si lo ha hecho, no ha participado un 100%, solo el tiempo que se ha necesitado en función al trabajo a realizar.

Gastos de personal imputables al proyecto

De manera similar al apartado anterior, en la siguiente tabla figuran los costes por perfil y actividad que se cargarán al proyecto. Todas las cantidades vienen expresadas en euros.

ACTIVIDAD	JP	DIS	ANA	PROG	R.PRU	TOTAL
ANÁLISIS DEL SISTEMA	1208,8	302,2	1284,4	0	0	2795,4
DISEÑO DEL SISTEMA	906,6	0	604,4	0	0	1511
IMPLEMENTACION DEL SISTEMA	0	0	604,4	6044	0	6648,4
PRUEBAS	0	0	0	483,52	4533	5016,5
IMPLANTACION DEL SISTEMA	0	0	453,3	453,3	0	906,6

Tabla 57: Gastos de personal de proyecto final

ABREVIATURA	PERFIL
JP	JEFE DE PROYECTO
DIS	DISEÑADOR SOFTWARE
ANA	ANALISTA SOFTWARE
PROG	PROGRAMADOR
R.PRU	RESPONSABLE DE PRUEBAS

Tabla 58: Perfiles que intervienen en el proyecto

Equipos y licencias software

DESCRIPCIÓN		COSTE (EUROS)
EQUIPOS		500,00 €
LICENCIAS SOFTWARE		250,00 €
	TOTAL	750,00 €

Tabla 59: Coste de equipos y licencias software

Material fungible y mantenimiento del local

DESCRIPCIÓN		COSTE (EUROS)
MATERIAL IMPRESO (PAPEL, TÓNER, ENCUADERNACIÓN,ETC.)		150,00 €
MATERIAL DE OFICINA		60,00 €
MANTENIMIENTO LOCAL		40,00 €
	TOTAL	250,00 €

Tabla 60: Costes de material fungible y mantenimiento del local

Resumen de costes del proyecto

DESCRIPCIÓN	COSTE (EUROS)
GASTOS DEL PERSONAL IMPUTABLES AL PROYECTO	16.877,87 €
EQUIPOS Y LICENCIAS SOFTWARE	750,00 €
MATERIAL FUNGIBLE Y GASTOS DEL LOCAL	250,00 €
OTROS GASTOS	0,00 €
TOTAL	17.877,87 €

Tabla 61: Resumen coste del presupuesto final

Resumen del presupuesto

El presupuesto final para el proyecto es el siguiente:

DESCRIPCIÓN	COSTE (EUROS)	
GASTOS DEL PERSONAL IMPUTABLES AL PROYECTO	16.877,87 €	
EQUIPOS Y LICENCIAS SOFTWARE	750,00 €	
MATERIAL FUNGIBLE	250,00 €	
SUBTOTAL	17.877,87 €	
COSTES INDIRECTOS	2.681,68 €	(15% SUBTOTAL)
RIESGO	3.375,57 €	(20% GASTOS PERSONAL)
BENEFICIO	1.817,18 €	(30% (COSTES INDIRECTOS + RIESGO))
TOTAL	25.752,30 €	

Tabla 62: Resumen del presupuesto final

Las fórmulas empleadas para el cálculo del presupuesto han sido las siguientes:

- Costes indirectos: se han calculado como el 15% de los costes directos imputados al proyecto.
- Riesgo: para calcular este importe, se ha aplicado un factor del 20% sobre los Gastos del personal, ya que es en este aspecto donde pueden sufrirse mayor número de variaciones.
- Beneficios: los beneficios se han calculado como el 30% de la suma de costes (directos e indirectos) y riesgo.

5.2.2 Planificación Final

Según se puede observar en la figura que representa el diagrama de Gantt de la planificación, la fecha de comienzo del proyecto se remonta a enero del 2009 y su finalización a mediados de octubre de ese mismo año.

La entrega del proyecto se ha visto demorada por distintos motivos. El primero, la inclusión a última hora de nuevas funcionalidades que eran necesarias y ofrecían muchas oportunidades a la aplicación, por otro lado, el hecho de tener que emplear una tecnología nueva ha propiciado en gran medida la demora de la entrega del sistema.

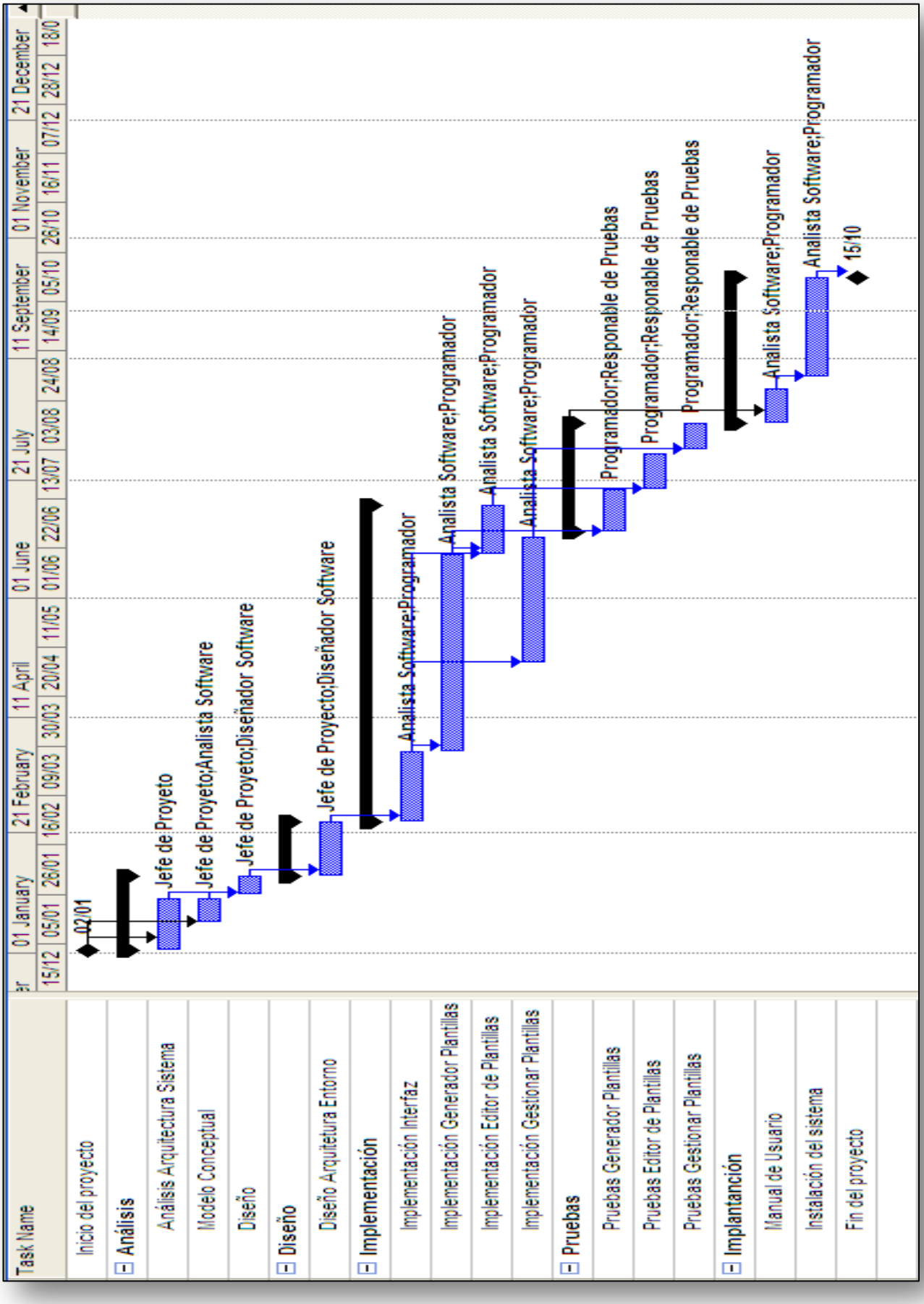


Ilustración 12: Diagrama Gantt. Planificación Final.

6 Análisis y Diseño (Desarrollo de la solución)

6.1 Descripción general

6.1.1 Perspectiva del producto

Actualmente, existe toda una gama de productos software para la gestión y desarrollo de proyectos informáticos, ya que es un sector demandado que es necesario debido a su importancia en muchas de las empresas que actualmente se encuentran en el mercado. Las funcionalidades que ofrecen estos productos software son muy amplias, pues suponen controlar las etapas de análisis y diseño dentro del ciclo de vida de un desarrollo software, además de toda la información ajena al desarrollo de la aplicación pero que resulta de gran importancia para alcanzar los objetivos marcados. En un principio, el producto está diseñado para formar parte de la herramienta SwReuser, pero dado el interés que suscita esta área, puede que la herramienta sea útil para dar apoyo a otras aplicaciones software que no incluyan este tipo de gestión del proyecto.

La funcionalidad de la herramienta ofrecida, se centra en los desarrollos de aplicaciones software, en cómo manejar todos los activos software tan importantes para la gestión del proyecto. Hasta el momento, las plantillas que se generan solo contemplan parte de los conceptos que recoge UML, por tanto se podría esperar que en un futuro abarcara más conceptos UML, tal vez se podrían incluir informes sobre la situación del proyecto, los cuales podrían ser de bastante utilidad para el jefe de proyecto. Inicialmente se pensó en una herramienta que solo abarcara la gestión de un único proyecto, posteriormente se barajo la posibilidad de guardar las plantillas software generadas con el propósito de ampliar la herramienta en sus funciones, pudiendo gestionar varios proyectos a la vez, lo cual resulta bastante útil para proyectos dependientes entre sí, cuya información se encuentra relacionada y de este modo se conseguiría un control exhaustivo de la información.

6.1.2 Capacidades generales

El producto software permite manejar toda la información relevante ajena al desarrollo software, ofreciendo la posibilidad de que el jefe de proyecto esté al tanto de todo lo que pueda ocurrir y ocurra en el proyecto. También ofrece una excelente comunicación entre el analista o analistas software y el jefe de proyecto, ya que dado que el analista software tiene el trato más directo y continuo con la plantilla de trabajo podrá estar pendiente de cualquier detalle que sea importante revisar, ya que tal vez conlleve un riesgo o necesite el chequeo a partir de una serie de pruebas, etc. Todo esto, quedará plasmando en las plantillas generadas por el sistema, pudiendo llevar un control exhaustivo del proyecto en cuestión.

6.1.3 Restricciones generales

Las pocas restricciones generales de este software están relacionadas con el acceso seguro al sistema y con el diagrama de elementos a generar.

Debe incluirse para este acceso, una ventana para la autenticación del cliente y así evitar el uso del sistema por parte de una persona no autorizada. Dado que este sistema será utilizado en el sector de desarrollo de proyectos informáticos, esta autenticación se realizará mediante el usuario y el password.

Para la herramienta, debe de haber como mínimo dos perfiles de usuario, uno de los cuales sería el jefe de proyecto, cuya función es establecer el diagrama de elementos, sobre el que se va a organizar toda la información con una aplicación ajena a la diseñada y generar las plantillas de trabajo a modo de comprobación y el otro acceso sería para el analista software que trabajará con las plantillas generadas a partir del diagrama de elementos gestionando toda la información que vaya surgiendo en el desarrollo del proyecto.

En cuanto, al diagrama de elementos, el jefe de proyecto va a tener una serie de limitaciones a la hora de generar el diagrama. En primer lugar, solo podrá establecer dos tipos de relaciones entre los elementos generados para que puedan ser leídos por la aplicación, estos son: Asociación y Herencia. En un futuro ó si el tiempo lo permite, no se descartará la idea de ampliar esta parte teniendo en cuenta más conceptos UML.

Por otro lado, aunque también relacionado con el diagrama de elementos, se ha restringido el número de elementos a introducir, es decir el usuario podrá seleccionar entre elementos establecidos por la herramienta. Estos elementos son: Requisito, Caso de Uso, Prueba, Riesgo y Gestión de configuración (versión). Además de estos, habrá otro elemento base que contiene información general del proyecto, que se denominará Proyecto.

En principio estas son las restricciones inicialmente plasmadas en el sistema desarrollado.

6.1.4 Características de los usuarios

Los usuarios dentro de la disciplina de la ingeniería del software que van a utilizar esta herramienta son:

Jefe de Proyecto: Es la figura clave en la planificación, ejecución y control del proyecto y es el motor que ha de impulsar el avance del mismo mediante la toma de decisiones tendentes a la consecución de los objetivos. El jefe de proyecto tiene poder ejecutivo y autoridad para mandar y tomar decisiones dentro del ámbito y objetivos del proyecto, aunque tampoco tiene el poder absoluto, ya que se encuentra inmerso en la estructura y organización de la empresa.

La herramienta implementada ofrece al jefe de proyecto la posibilidad de gestionar los activos de su proyecto a través de un diagrama de elementos, donde identificará elementos relevantes para el proyecto en cuestión, señalando sus atributos importantes y estableciendo las relaciones que él crea conveniente. De este modo, fijara en cierta medida sus intereses con respecto al proyecto y a la consecución de los objetivos. Además, dichos objetivos podrán ser chequeados por el Jefe de Proyecto a través de la generación de plantillas software, comprobando las posibles carencias y virtudes del diagrama de elementos.

Analista Software: Es aquel individuo con capacidad de abstracción que ejerce las tareas de análisis de los sistemas informáticos, con el fin de automatizarlos. El analista tiene por cometido estudiar en profundidad un problema y describirlo con el propósito de ser solucionado mediante un sistema informático.

La herramienta a desarrollar permite al analista resolver sus funciones hacia el resultado deseado, ya que podrá gestionar las plantillas en base al trabajo realizado por el jefe de proyecto en el paso previo a la creación de dichas plantillas. El analista podrá gestionar todos los activos que se consideren de interés, de modo que pueda determinar el comportamiento del software y la garantía de calidad para garantizar las expectativas del cliente.

Aunque solo se han señalado estos dos usuarios como los más significativos a la hora de utilizar el sistema, puede que el equipo de desarrollo restante, (diseñadores, programadores, ingenieros de pruebas, etc.) participen en el uso y gestión de las plantillas, dependerá en parte de la jerarquía establecida dentro del equipo de trabajo, cuestión ajena al desarrollo de la aplicación.

6.1.5 Suposiciones y dependencias

Se supone que todos los equipos donde va a ser instalada la herramienta disponen de todo el hardware y software necesario para su uso.

Se asume que el diseño de SwReuser ha sido realizado previamente y que por este motivo cumple con todos los requisitos necesarios para su funcionamiento.

El lenguaje utilizado por el sistema será el castellano.

6.2 Modelo conceptual

El diagrama de clases es el diagrama principal de diseño y análisis para el sistema. En él, la estructura de clases del sistema se especifica, con relaciones entre clases y estructuras de herencia, que en este caso no es necesario emplear. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal, modelando el dominio de la aplicación con un alto nivel de abstracción. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.

En la siguiente figura se muestra el diagrama de clases de diseño realizado durante la fase de análisis de la aplicación:

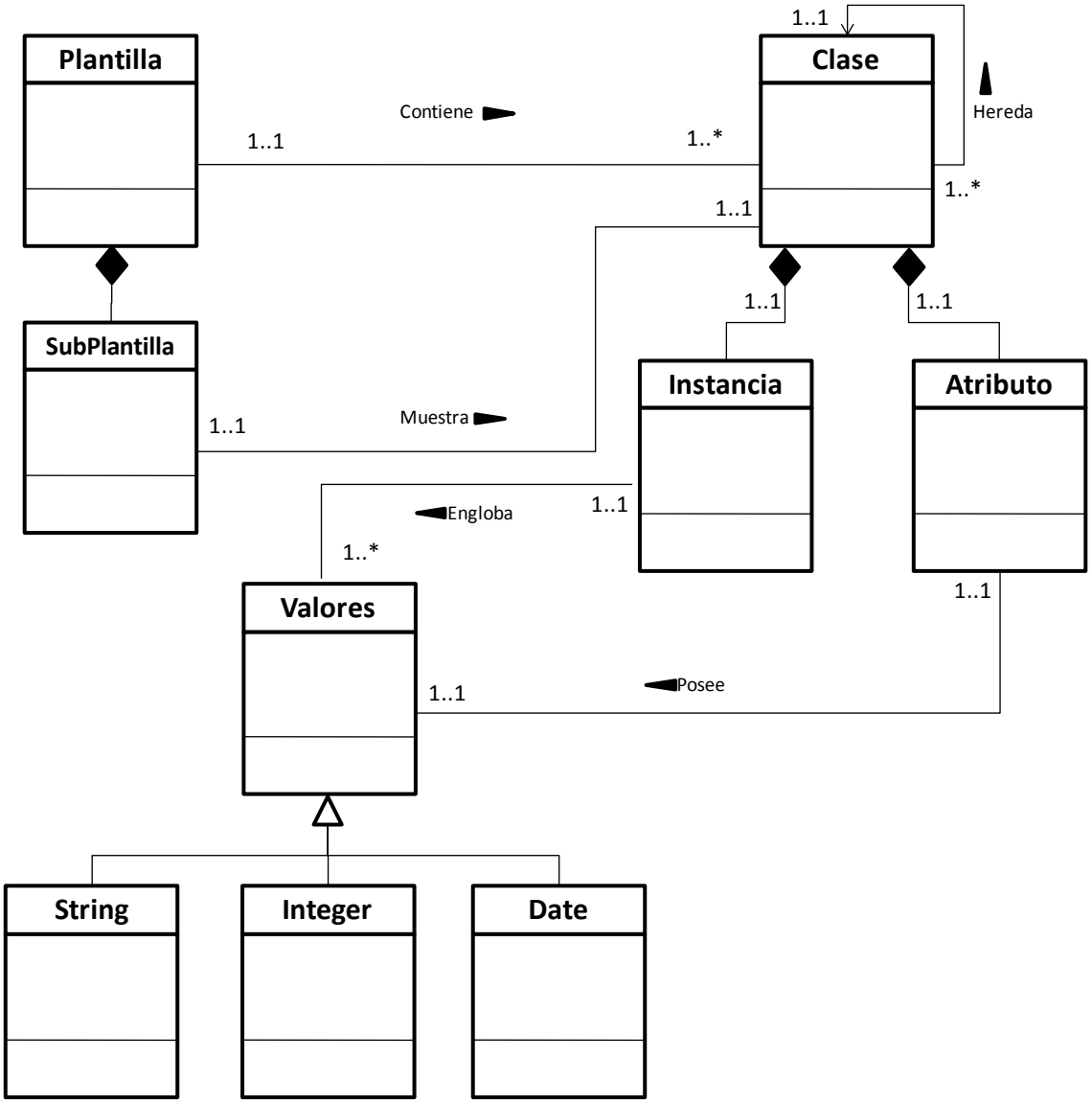


Ilustración 13: Modelo Conceptual

6.2.1 Identificación de Responsabilidades

NOMBRE	Plantilla
DESCRIPCIÓN	Esta clase representa una plantilla donde se puede visualizar cada una de las clases del diagrama de elementos.

Tabla 63: Clase Plantilla

NOMBRE	SubPlantilla
DESCRIPCIÓN	Esta clase representa la información asociada a una Clase concreta para un diagrama de elementos específico.

Tabla 64: Clase SubPlantilla

NOMBRE	Clase
DESCRIPCIÓN	Esta clase representa una Clase generada por el Jefe de Proyecto al dibujar el diagrama de elementos.

Tabla 65: Clase Clase

NOMBRE	Instancia
DESCRIPCIÓN	Esta clase representa cada uno de los registros de información que conforma una clase.

Tabla 66: Clase Instancia

NOMBRE	Atributo
DESCRIPCIÓN	Esta clase representa cada uno de los atributos de una clase concreta.

Tabla 67: Clase Atributo

NOMBRE	Valores
DESCRIPCIÓN	Esta clase representa el valor de un atributo para una instancia determinada.

Tabla 68: Clase Valores

NOMBRE	String
DESCRIPCIÓN	Esta clase representa el valor de tipo cadena de un atributo para una instancia determinada.

Tabla 69: Clase String

NOMBRE	Integer
DESCRIPCIÓN	Esta clase representa el valor de tipo numérico de un atributo para una instancia determinada.

Tabla 70: Clase Integer

NOMBRE	Date
DESCRIPCIÓN	Esta clase representa el valor de tipo fecha de un atributo para una instancia determinada.

Tabla 71: Clase Date

6.2.2 Identificación de Generalidades

NOMBRE	Valores
DESCRIPCIÓN	Esta generalidad representa el hecho de que un valor puede ser de tipo string, integer o date.

Tabla 72: Generalización 1

6.2.3 Identificación de Asociaciones y Agregaciones

Plantilla → Clase (Contiene)	
TIPO	Asociación
CARDINALIDADES	- Plantilla (1..1) - Clase (1..*)
DESCRIPCIÓN	- Una plantilla contiene al menos una sola clase. - Una clase forma parte de una única plantilla.

Tabla 73: Asociación 1

Plantilla → SubPlantilla	
TIPO	Composición
CARDINALIDADES	- Plantilla (1..1) - SubPlantilla (1..*)
DESCRIPCIÓN	- Una plantilla está compuesta al menos por una subplantilla. - Una subplantilla pertenece a una y solo una plantilla.

Tabla 74: Composición 1

SubPlantilla → Clase (Muestra)	
TIPO	Asociación
CARDINALIDADES	- SubPlantilla (1..1) - Clase (1..1)
DESCRIPCIÓN	- Una subplantilla muestra una y solo una clase. - Una clase es mostrada en una y solo una subplantilla.

Tabla 75: Asociación 2

Clase → Clase	
TIPO	Asociación
CARDINALIDADES	- Clase (1..1) - Clase (1..*)
DESCRIPCIÓN	- Una clase tiene una sola clase madre. - Una clase madre tiene 1 o varias clases hijas.

Tabla 76: Asociación 3

Clase → Instancia	
TIPO	Composición
CARDINALIDADES	- Clase (1..1) - Instancia (1..*)
DESCRIPCIÓN	- Una clase está compuesta por al menos una instancia. - Una instancia pertenece a una y solo una clase.

Tabla 77: Composición 2

Clase → Atributo	
TIPO	Composición
CARDINALIDADES	- Clase (1..1) - Atributo (1..*)
DESCRIPCIÓN	- Una clase está compuesta por al menos un atributo. - Un atributo pertenece a una y solo una clase.

Tabla 78: Composición 3

Instancia → Valores (Engloba)	
TIPO	Asociación
CARDINALIDADES	- Instancia (1..1) - Valores (1..*)
DESCRIPCIÓN	- Una instancia engloba al menos un valor. - Un valor es englobado a una y solo una instancia.

Tabla 79: Asociación 4

Atributo → Valores (Posee)	
TIPO	Asociación
CARDINALIDADES	- Atributo (1..1) - Valores (1..1)
DESCRIPCIÓN	- Un valor pertenece a uno y solo un atributo. - Un atributo posee solo un valor.

Tabla 80: Asociación 5

Valores → String	
TIPO	Generalización
CARDINALIDADES	
DESCRIPCIÓN	- El valor de tipo string es un valor.

Tabla 81: Generalización 1

Valores → Integer	
TIPO	Generalización
CARDINALIDADES	
DESCRIPCIÓN	- El valor de tipo integer es un valor.

Tabla 82: Generalización 2

Valores → Date	
TIPO	Generalización
CARDINALIDADES	
DESCRIPCIÓN	- El valor de tipo date es un valor.

Tabla 83: Generalización 3

6.3 Diseño del sistema

Este apartado tiene como propósito describir el diseño a alto nivel seguido en la aplicación a desarrollar, creando un marco de trabajo sobre el diseño, decidiendo los patrones arquitectónicos a seguir.

Dicho documento se centrará en los equipos de desarrollo software, pero sin dejar de lado a los usuarios, ingenieros de sistemas y hardware, además del personal de mantenimiento.

El contexto en el que se va a desarrollar el sistema es el siguiente:

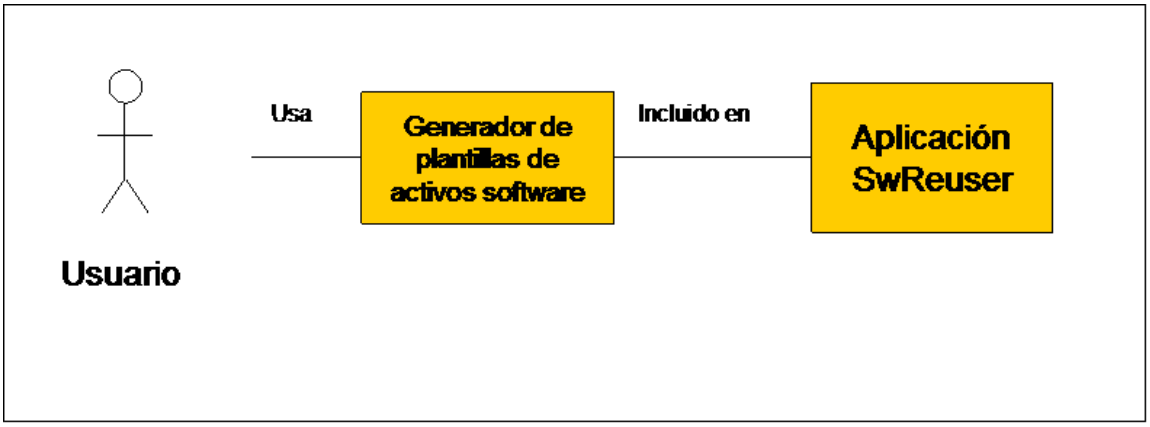


Ilustración 14: Diseño del sistema

Como se puede apreciar en la figura anterior, el sistema de acceso a la aplicación utilizado es monousuario, en el que se proveen servicios a un solo usuario en un determinado tiempo. De este modo, este módulo integrado en la aplicación SwReuser estará en un solo equipo, el cual ofrecerá la posibilidad de ejecutar las distintas funcionalidades del sistema. Además el acceso a la aplicación podrá realizarse únicamente desde el equipo en el que se encuentre instalado, de este modo no existirá ningún problema para ninguno de los usuarios que acceden, ya que el acceso entre usuarios no podrá ser simultáneo, como ocurre en los sistemas multiusuario.

El usuario accede al sistema mediante una interfaz gráfica, que mostrará por pantalla tanto el menú de opciones como las distintas funcionalidades junto con el resultado obtenido (plantillas de trabajo, archivo con extensión .xml). Para el establecimiento de eventos y escenarios, simplemente el usuario deberá elegir dicha opción del menú de opciones, accediendo a las vistas correspondientes donde podrá acceder a las funcionalidades que desea ejecutar.

6.3.1 Identificación de mecanismos genéricos de diseño

Los estilos de arquitectura propuestos por Shaw y Garlan en 1996, definen el concepto estilo, como una familia de sistemas en términos de un patrón de organización estructural. Un estilo define un vocabulario de tipos de componentes y conectores, restricciones de combinación y uno o más modelos semánticos para determinar propiedades del todo a partir de las partes.

En el diseño del sistema se han empleado patrones de diseño arquitectónicos como mecanismos genéricos de resolución de problemas. En este apartado se van a explicar los patrones arquitectónicos en que está basada la solución que se ha adoptado.

El método de diseño escogido para la realización de la aplicación ha sido una **arquitectura basada en el Modelo Vista Controlador (MVC)**, debido a la propia naturaleza de la aplicación, ya que se puede apreciar fácilmente una clara diferencia entre los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Desarrollar una aplicación siguiendo este patrón de diseño tiene muchas ventajas:

- La aplicación está implementada modularmente.
- Sus vistas muestran información actualizada siempre.
- El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Si se desea hacer una modificación del dominio, como aumentar métodos o datos contenidos, sólo debe modificarse el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
- Las modificaciones a las vistas no afectan en absoluto a los otros módulos de la aplicación.
- MVC es bastante utilizado en la actualidad en marcos de aplicación orientados a objeto desarrollados para construir aplicaciones de gran tamaño; Java Swing, Apache Struts, Microsoft ASP.NET, las transformaciones XSL, etc.

- MVC está demostrado ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

Pero, también tiene una serie desventajas:

- El tiempo de desarrollo de una aplicación que implementa el patrón de diseño MVC es mayor, al menos en la primera etapa, que el tiempo de desarrollo de una aplicación que no lo implementa, ya que MVC requiere que el programador implemente una mayor cantidad de clases que en un entorno de desarrollo común no son necesarias. Sin embargo, esta desventaja es muy relativa ya que posteriormente, en la etapa de mantenimiento de la aplicación, una aplicación MVC es muchísimo más mantenible, extensible y modificable que una aplicación que no lo implementa.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación. Esta arquitectura inicial debe incluir, por lo menos: un mecanismo de eventos para poder proporcionar las notificaciones que genera el modelo de aplicación; una clase Modelo, otra clase Vista y una clase Controlador genéricas que realicen todas las tareas de comunicación, notificación y actualización que serán luego transparentes para el desarrollo de la aplicación.
- MVC es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

La arquitectura empleada se divide:

Vista: En esta residen todas las clases dedicadas a la presentación de datos de cara al usuario; siendo esta la que se encargue de interaccionar con el usuario, definida coloquialmente como la interfaz de usuario.

Controlador: En la cual residen las clases encargadas de recibir e interpretar la interacción del usuario, actuando sobre los diferentes modelos y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como de su visualización. Además, estará relacionada con el Gestor de Datos, la cual estará encargada de la persistencia de los datos.

Modelo: En esta residen todas las clases que contienen la lógica de negocio. Cada uno de los tres modelos con los que trabaja la aplicación recibirá las peticiones enviadas por el controlador, procesará las peticiones y le devolverá al controlador de información el resultado para que sea presentado al usuario a través de la vista.

Gestor de datos: Reúne las clases encargadas de comunicarse con los ficheros de información, tanto los ficheros con extensión .csem que contienen el diagrama de elementos como los .xml generados por la aplicación para guardas las plantillas generadas.

6.4 Diseño de la arquitectura de módulos del sistema

En este apartado se especificarán los distintos módulos de los que consta el sistema.

6.4.1 Módulos del sistema

A lo largo de este apartado se especificarán los distintos módulos de los que está compuesto el sistema.

En la siguiente ilustración se muestra un diagrama que explica el conjunto de componentes que conforman el sistema. Este diagrama está expresado a un alto nivel, a medida que vayamos avanzando en este apartado iremos bajando en nivel de abstracción.

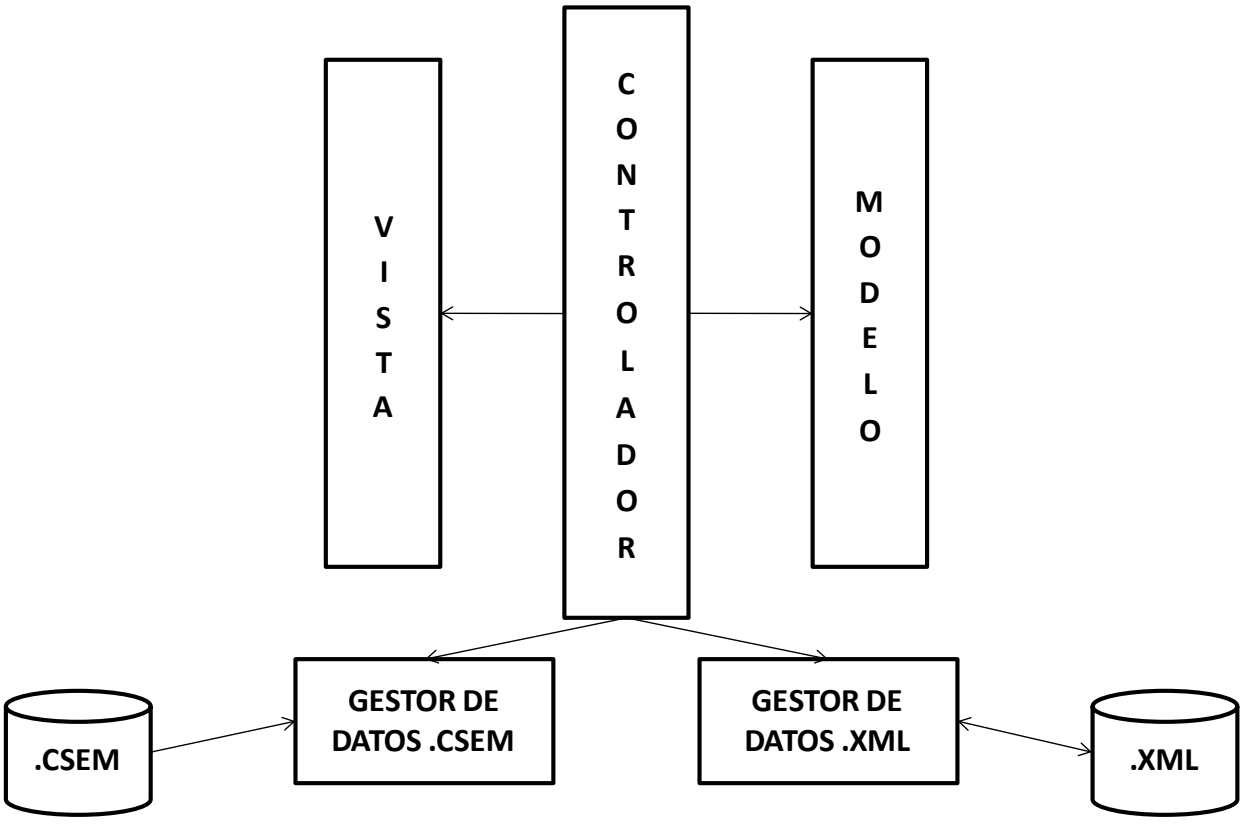


Ilustración 15: Módulos del Sistema

Según se observa en la figura, los ficheros con extensión .csem y .xml están al mismo nivel, esto es debido a que son componentes similares que contienen datos significativos para la aplicación. El gestor de datos se encarga de llevar a cabo el tratamiento de dichos datos para poder proporcionárselo al componente Controlador.

El cometido de este apartado es centrarnos en cada uno de estos componentes:

- Vista
- Controlador de información
- Modelo
- Gestor de datos

Como se puede observar en la figura siguiente, el controlador de información es el componente con mayor número de responsabilidades, ya que se encarga de comunicarse con la vista, el modelo y el gestor de datos. El controlador de información se comunica con el resto de componentes a través de las interfaces proporcionadas por los mismos. Con el objeto de optimizar la legibilidad del diagrama, se han simplificado las interfaces empleando interfaces genéricas que representan las diferentes de las clases que contienen los componentes.

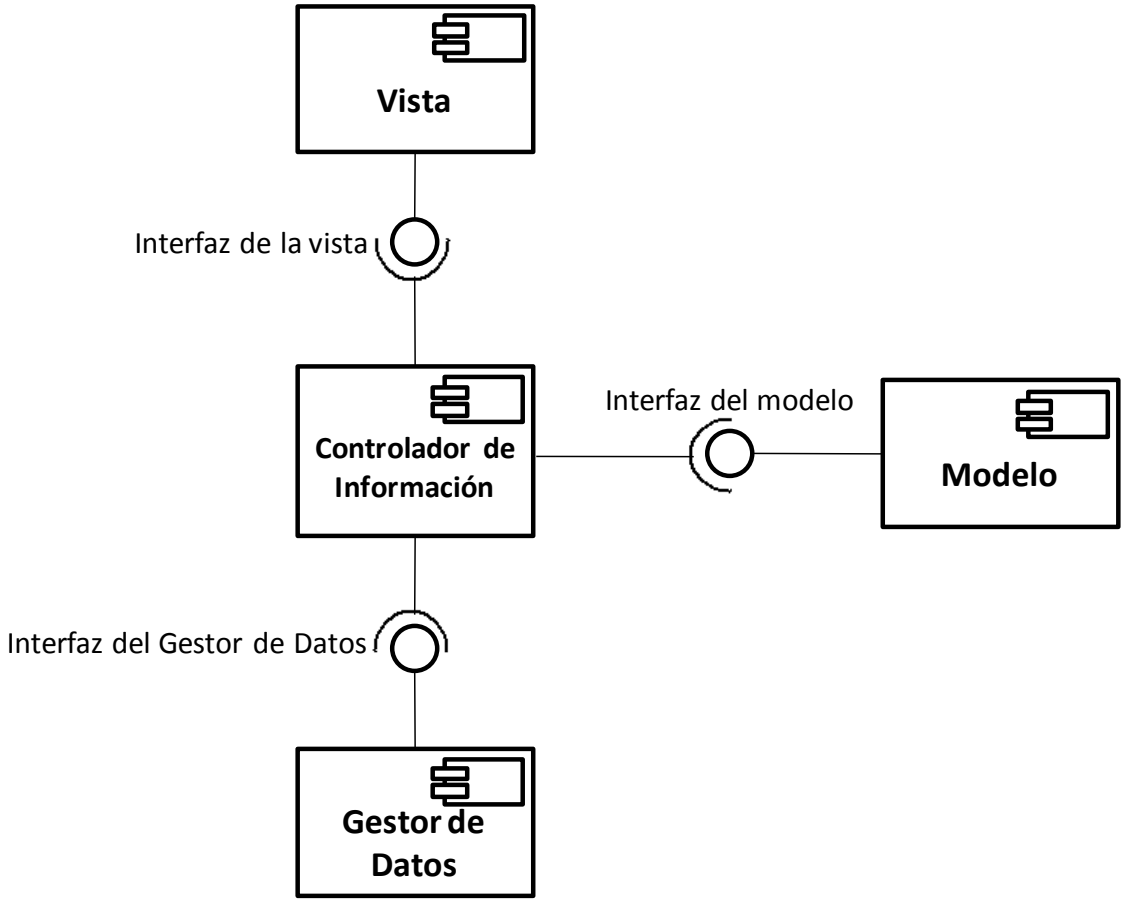


Ilustración 16: Diagrama de componentes

6.4.2 Descomposición en Módulos

En este apartado se detallarán cada uno de los módulos de los que está compuesto el sistema.

6.4.2.1 Vista

Este módulo, como se indico anteriormente, será el encargado de interactuar con el usuario y de presentarle la información.

En este paquete incluimos todas aquellas clases que contienen una interfaz de trabajo para el usuario. De este modo, el usuario inicialmente se encontrará con una pantalla inicial con tres opciones posibles, cada una de las opciones tienen por finalidad la carga de archivos .csem (Loader) y .xml (Gestor Plantillas). Cada una de las cargas genera una plantilla de trabajo que será utilizada por el analista del proyecto.

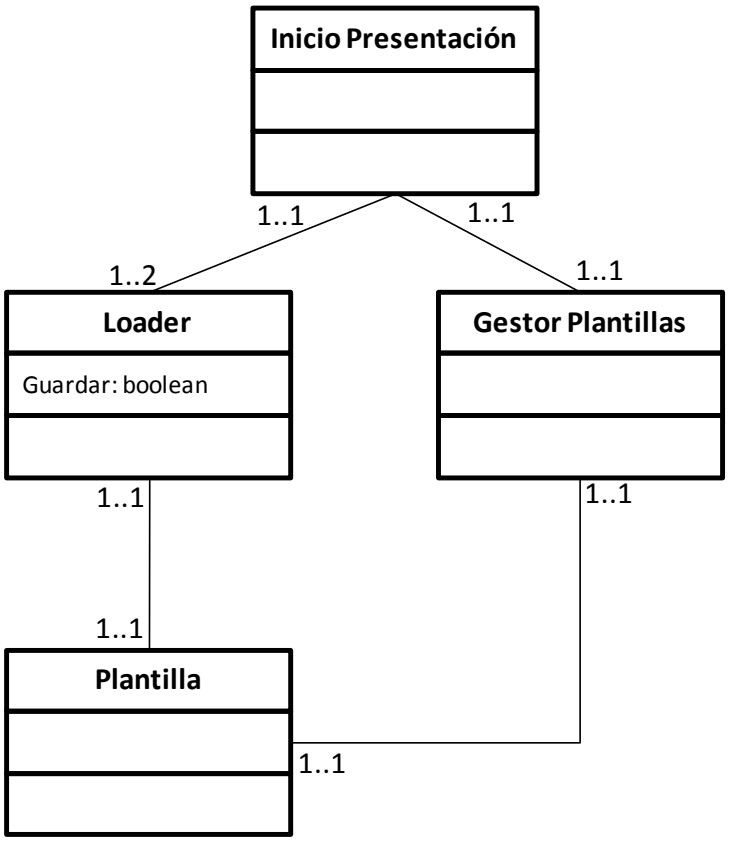


Ilustración 17: Vista del sistema

Para la interfaz de trabajo se utilizará una arquitectura de sistemas por eventos, ya que tanto en la gestión del diagrama de elementos como las plantillas de trabajo se tratan como un sistema de transición de estados, en el que cada una de las componentes espera hasta que ocurre un evento que la afecta.

6.4.2.2 Controlador de Información

Este módulo realiza el control interno del sistema, es decir, relaciona los distintos componentes: modelo, vista y gestor de datos. Este control soporta básicamente la evolución de la información utilizada en distintos modelos de trabajo. De este modo, partimos de un archivo .csem generado por un sistema ajeno (SwReuser) a nuestra aplicación, el cual será utilizado como fuente por la aplicación diseñada que leerá bajo un modelo Software Enginering, el cual será tratado por la aplicación transformándolo en el modelo diseñado como parte de la solución del sistema, el cual se puede visualizar en la parte de Análisis (Apartado 6.2.).

A continuación, este modelo será transformado en plantillas de trabajo, que serán tratadas por el usuario como medio para guardar información relevante, por último, estas plantillas serán tratadas para ser guardadas en un archivo específico bajo un modelo XML generado por el sistema.

Como se puede ver, la aplicación trata la información bajo distintos modelos y es mostrada bajo diferentes vistas.

6.4.2.3 Modelo de diseño

En este modulo se encuentran todas las clases que se encargan de la lógica de negocio del sistema.

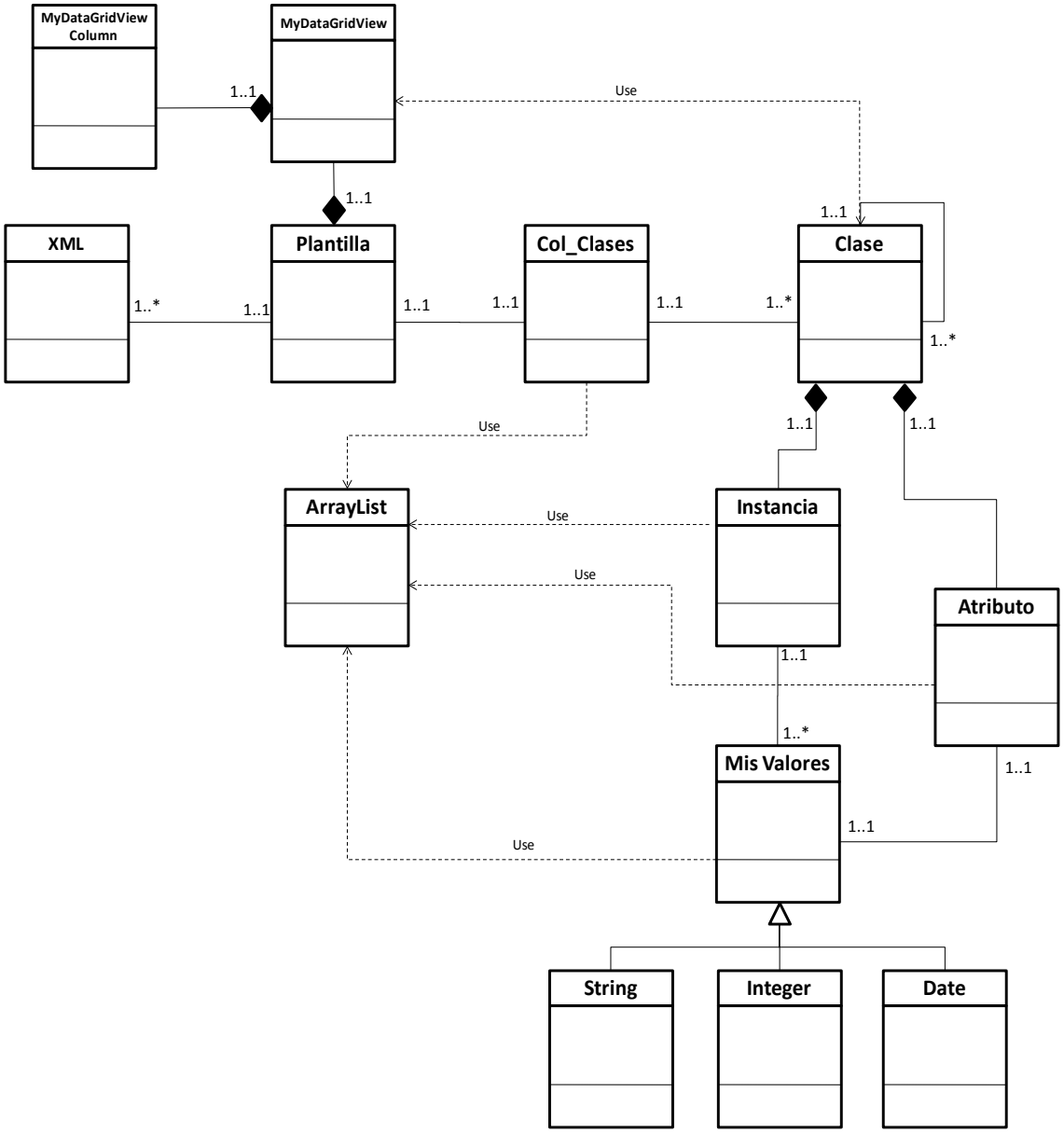


Ilustración 18: Modelo de Diseño del sistema

Si comparamos este diagrama con el propuesto en la fase de análisis, vemos que hay algunas clases que han desaparecido y otras que han sido añadidas. Las nuevas clases que han sido añadidas son estructuras de datos usadas para almacenar elementos tales con arraylist y datagridview.

Se han incluido como nuevas clases las siguientes:

NOMBRE	**MyDataGridView**
DESCRIPCIÓN	Esta clase representa cada una de las clases del diagrama de elementos visualmente sobre plantillas de trabajo.

Tabla 84: Clase MyDataGridView

NOMBRE	**MyDataGridViewColumn**
DESCRIPCIÓN	Esta clase representa cada atributo de cada una de las clases del diagrama de elementos, como columna del DataGridView.

Tabla 85: Clase MyDataGridViewColumn

NOMBRE	**ArrayList**
DESCRIPCIÓN	Esta clase representa conjunto de clases, instancias, atributos y valores.

Tabla 86: Clase ArrayList

NOMBRE	**XML**
DESCRIPCIÓN	Esta clase representa el conjunto de DataGridView bajo un archivo con formato XML.

Tabla 87: Clase XML

NOMBRE	**Col_Clases**
DESCRIPCIÓN	Esta clase representa como su nombre indica, el conjunto de clases que contiene el diagrama de elementos cargado por la aplicación.

Tabla 88: Clase Col_Clases

6.4.2.4 Gestor de Datos

En este módulo se encuentran todas las clases que gestionan los archivos .csem y .xml, es decir aquellos archivos destinados a guardar información relevante del proyecto en cuestión.

Los archivos .csem son generados por una aplicación (SwReuser) ajena a la diseñada, y la aplicación creada solo utilizará este tipo de archivos como fuente para la generación de nuestro modelo de datos y plantillas de trabajo. La aplicación en ningún momento insertará, modificará o eliminará datos de este archivo

Los archivos .xml son generados por la aplicación diseñada para recoger toda la información contenida en las plantillas. La información se encuentra jerarquizada de la siguiente manera:

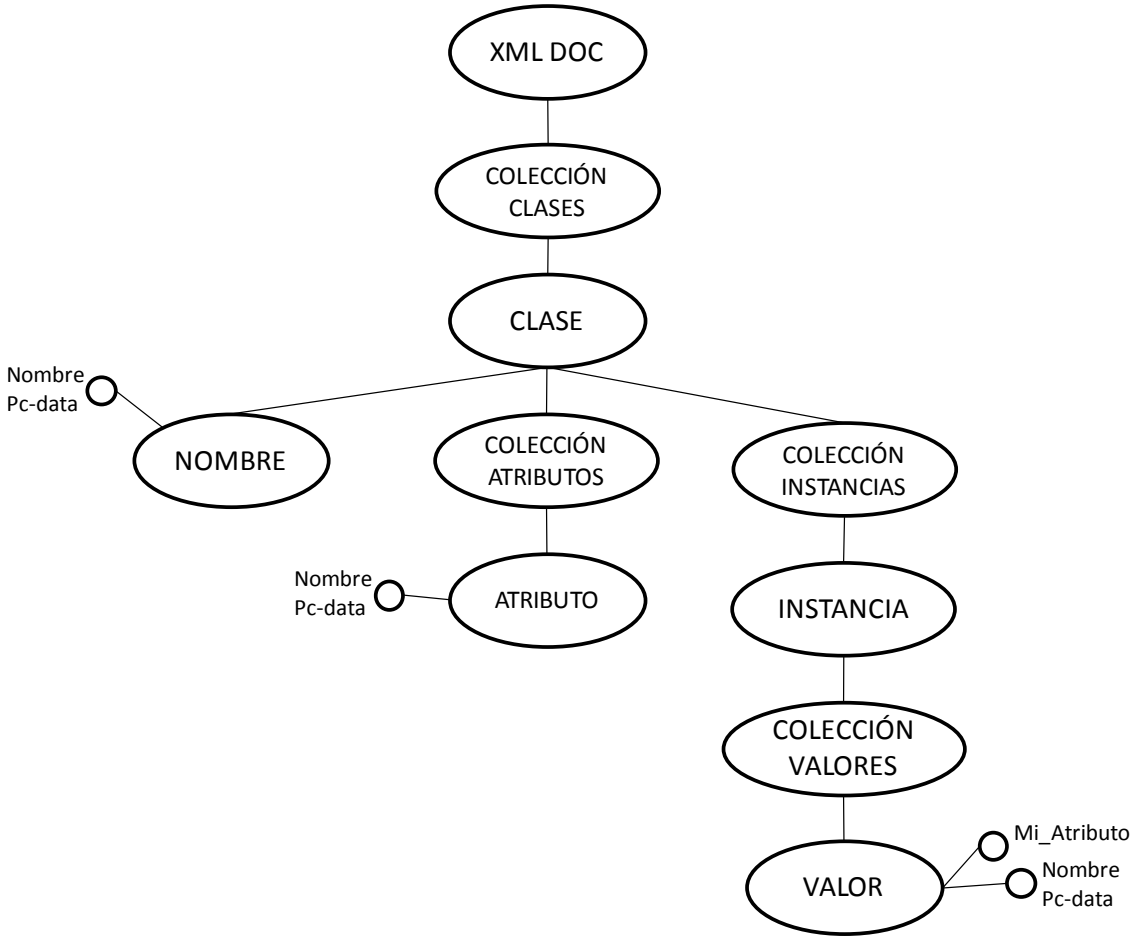


Ilustración 19: Modelo XML del sistema

Estos archivos aparte de ser creados por la aplicación, pueden ser recuperados generando nuevamente las plantillas de trabajo, cuyos datos podrán ser modificados o eliminados, siendo también posible la inserción de nueva información. Todos estos cambios serán guardados por el usuario en un nuevo archivo o bien reemplazando el archivo existente con las nuevas modificaciones.

Las clases encargadas en el tratamiento de ambos ficheros son:

- Loader: destinada a la carga de los archivos .csem.
- XML: destinada al tratamiento de archivos .xml.

7 Pruebas

En este apartado se especifican el conjunto de pruebas unitarias a las que se ha sometido a la aplicación.

En primer lugar se han realizado una serie de pruebas para comprobar si es correcta la generación de plantillas. Si recupera todos los datos requeridos: clases, atributos, herencias,… Para ello, se han tenido en cuenta 5 casos de pruebas que son los siguientes:

Prueba 1.

El diagrama de elementos dibujado a continuación representa un pequeño caso de pruebas, compuesto por tres clases, dos de ellas relacionadas bajo una asociación y otras dos bajo una herencia. Se ha de tener en cuenta que la plantilla generada por la Clase3 debe tener además de sus atributos, los atributos de la Clase1, debido a la relación de herencia establecida entre ellas.

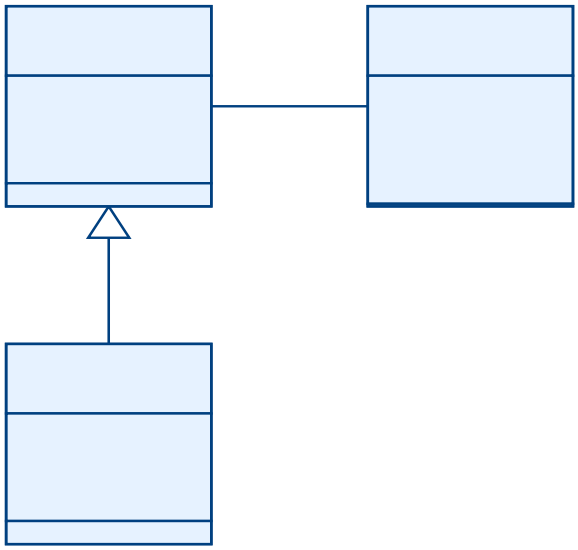


Ilustración 20: Diagrama de Clases. Prueba 1

Resultado de la prueba 1.

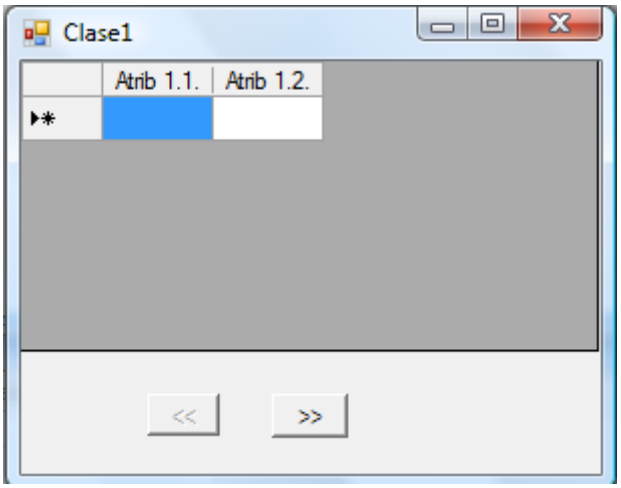


Ilustración 21: Resultado 1.1. Prueba 1

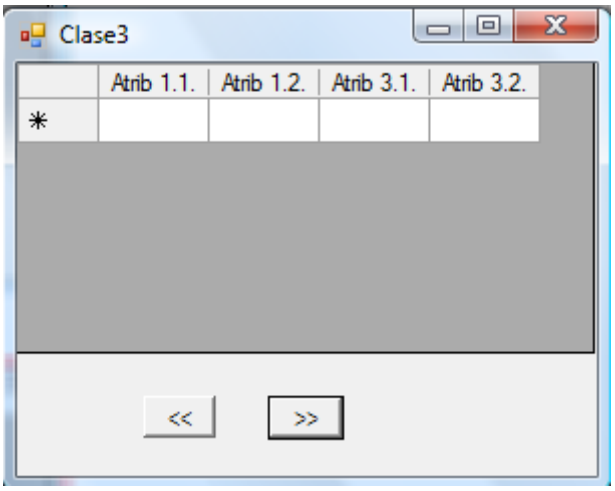


Ilustración 22: Resultado 1.2. Prueba 1

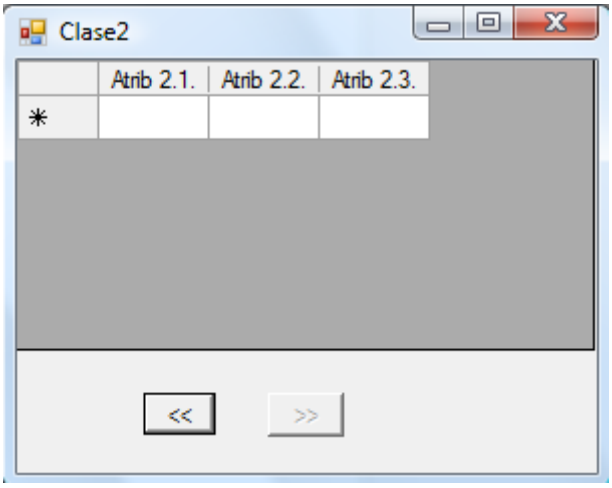


Ilustración 23: Resultado 1.3. Prueba 1

Prueba 2.

El diagrama de elementos dibujado a continuación representa un pequeño caso de pruebas, compuesto por cuatro clases, dos de ellas relacionadas bajo una asociación y otras tres bajo una herencia. Se ha de tener en cuenta que la plantilla generada por la Clase3 y Clase4 debe tener además de sus atributos, los atributos de la Clase1, debido a la relación de herencia establecida entre ellas.

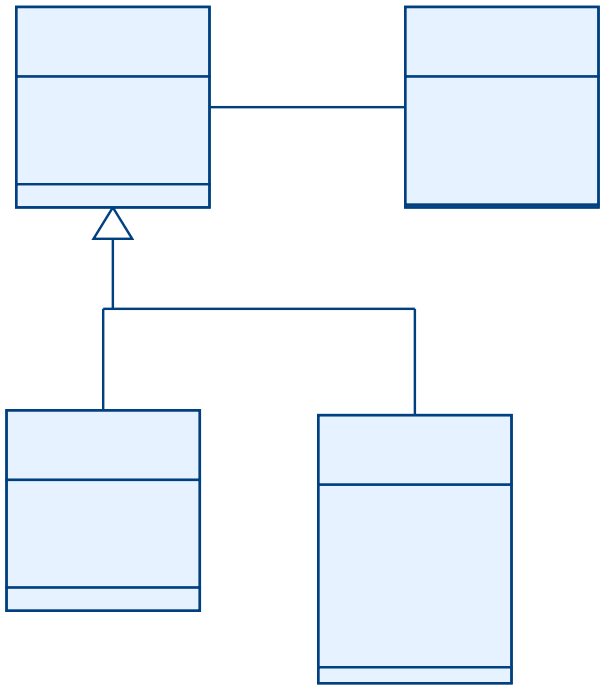


Ilustración 24: Diagrama de clases. Prueba 2

Resultado de la prueba 2.

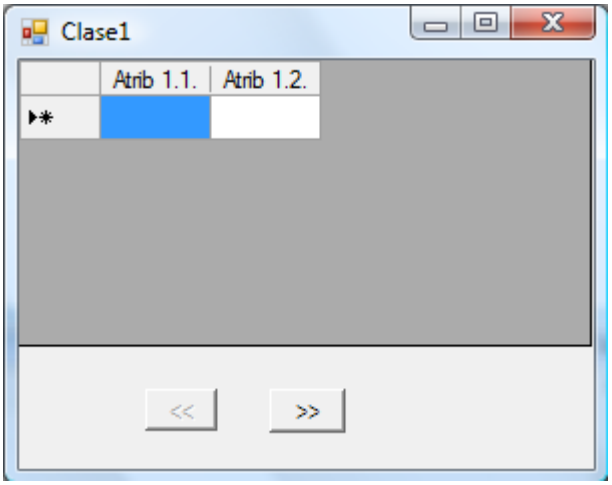


Ilustración 25: Resultado 2.1. Prueba 2

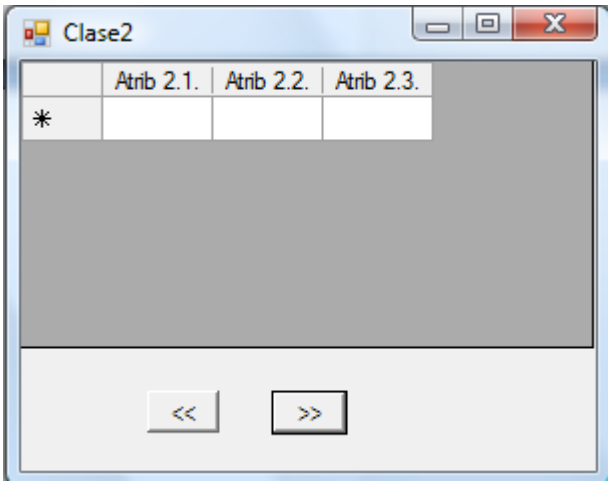


Ilustración 26: Resultado 2.2. Prueba 2

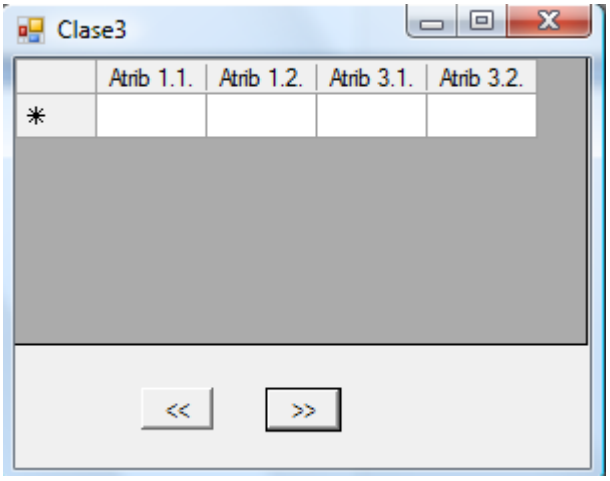


Ilustración 27: Resultado 2.3. Prueba 2

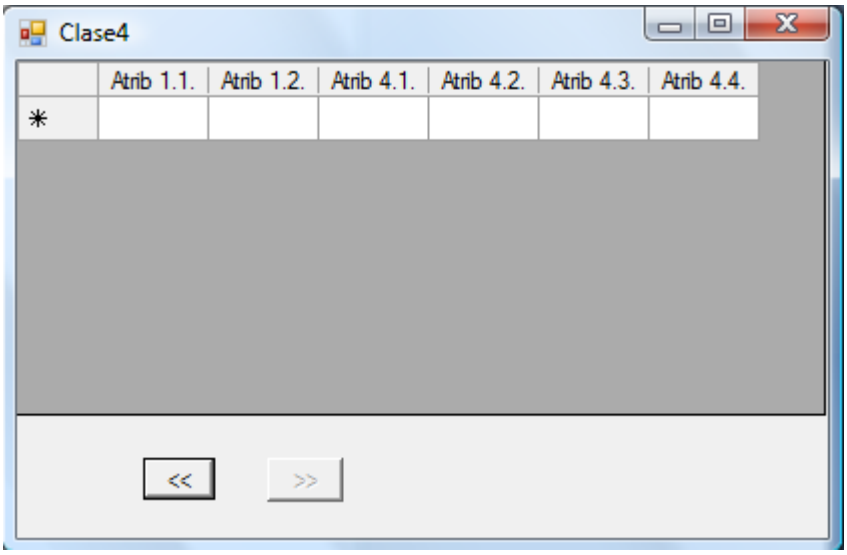


Ilustración 28: Resultado 2.4. Prueba 2

Prueba 3.

El diagrama de elementos dibujado a continuación representa un caso de pruebas, compuesto por cinco clases, relacionadas mediante asociación y herencia. Se ha de tener en cuenta que la plantilla generada por la Clase3 y Clase4 debe tener además de sus atributos, los atributos de la Clase1, debido a la relación de herencia establecida entre ellas; lo mismo ocurre con la Clase5 con respecto a las Clase3.

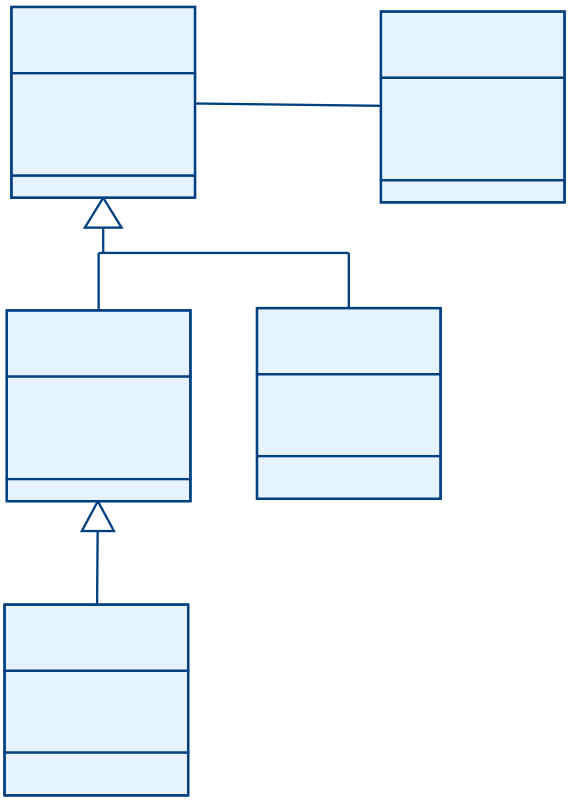


Ilustración 29: Diagrama de clases. Prueba 3

Resultado de la prueba 3.

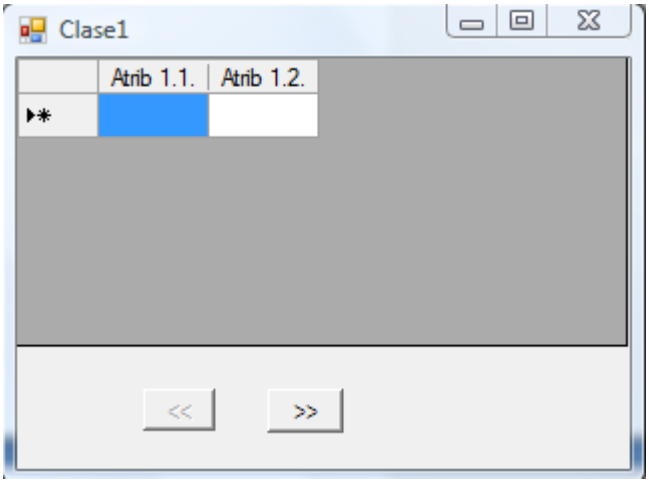


Ilustración 30: Resultado 3.1. Prueba 3

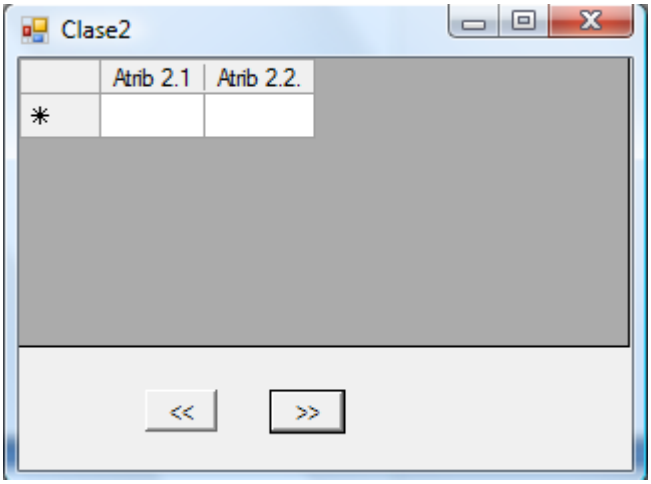


Ilustración 31: Resultado 3.2. Prueba 3

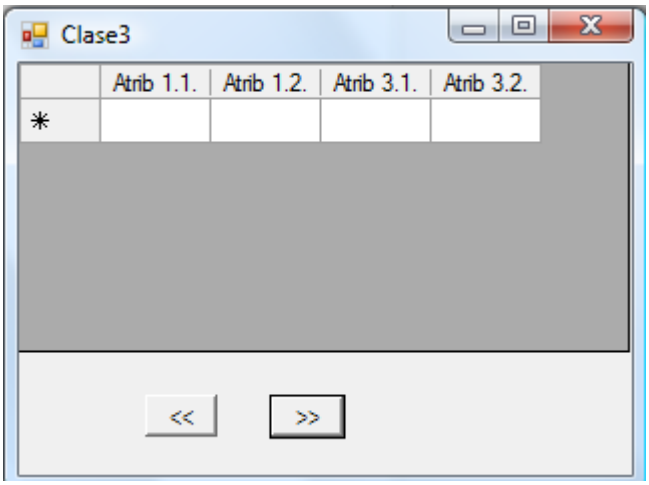


Ilustración 32: Resultado 3.3. Prueba 3

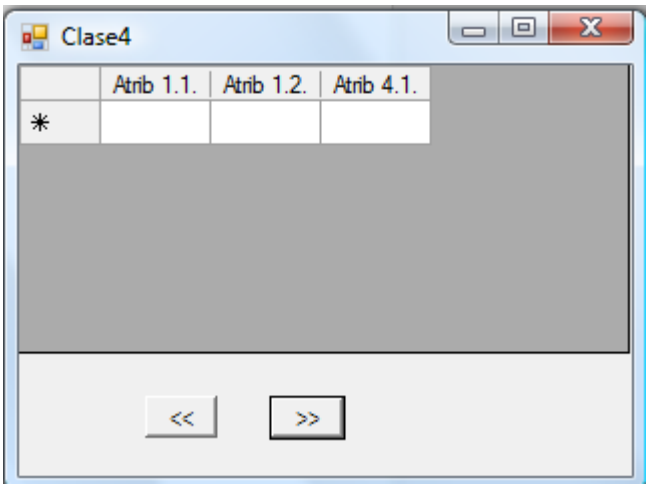


Ilustración 33: Resultado 3.4. Prueba 3

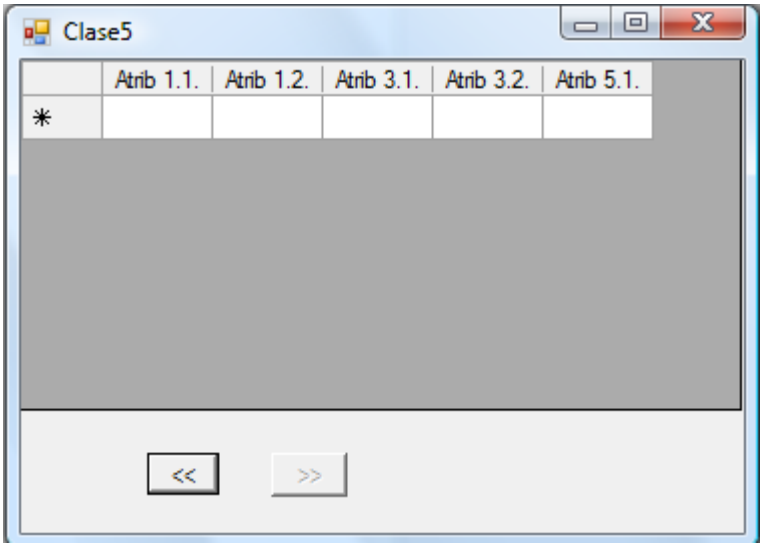


Ilustración 34: Resultado 3.5. Prueba 3

Prueba 4.

El diagrama de elementos dibujado a continuación representa un caso de pruebas, compuesto por seis clases, relacionadas bajo relaciones de herencia y asociación. Se ha de tener en cuenta que la plantilla generada por la Clase3 y Clase4 debe tener además de sus atributos, los atributos de la Clase1, debido a la relación de herencia establecida entre ellas; lo mismo ocurre con la Clase5 y Clase6 con sus respectivas clases Clase3 y Clase5. Además se tienen en cuenta aquellos atributos repetidos en diferentes clases relacionadas mediante herencia, para estos atributos solo aparecerán una por clase y plantilla, como se puede apreciar en las plantillas.

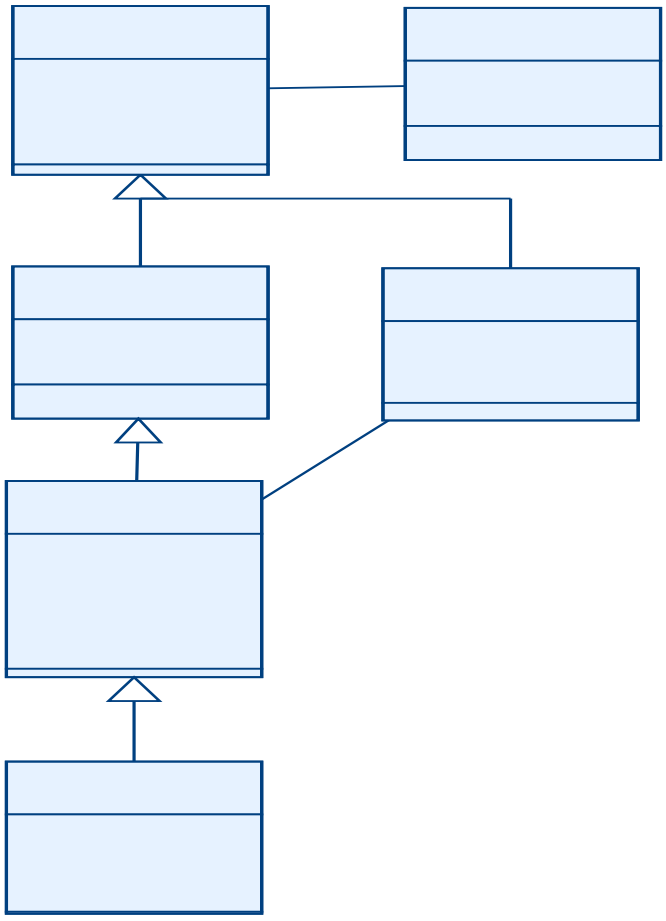


Ilustración 35: Diagrama de clases. Prueba 4

Resultado de la Prueba 4.

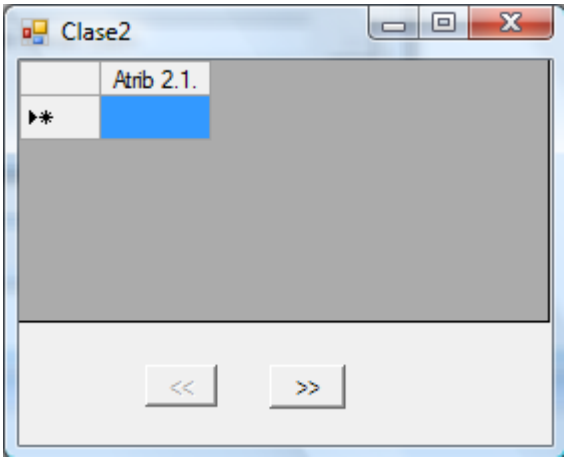


Ilustración 36: Resultado 4.1. Prueba 4

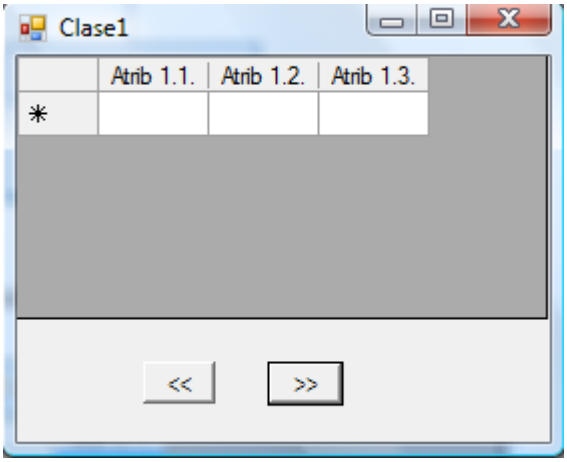


Ilustración 37: Resultado 4.2. Prueba 4

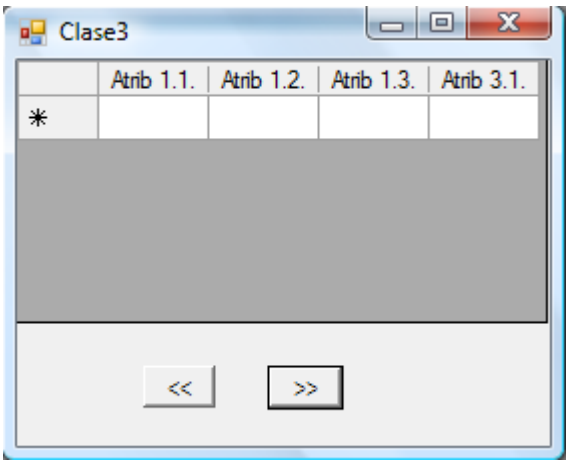


Ilustración 38: Resultado 4.3. Prueba 4

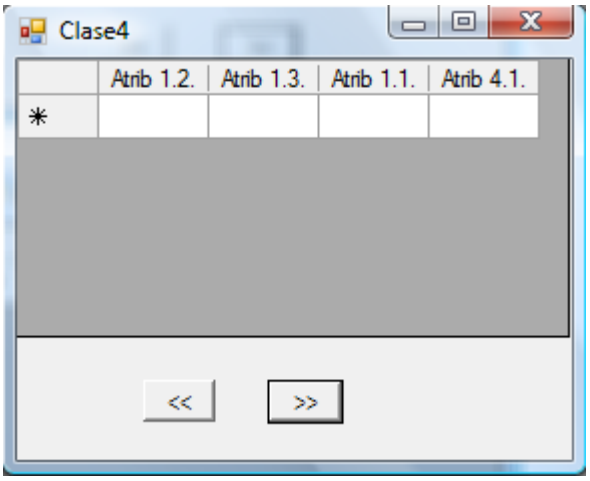


Ilustración 39: Resultado 4.4. Prueba 4

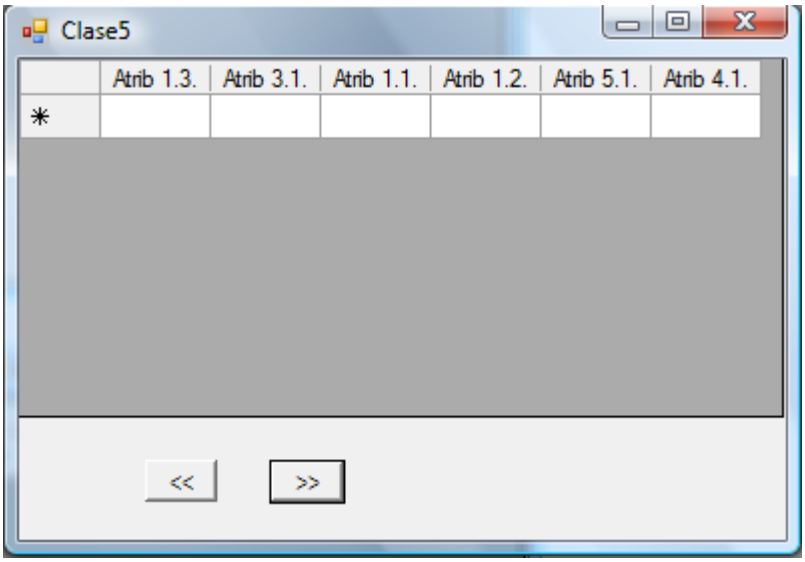


Ilustración 40: Resultado 4.5. Prueba 4

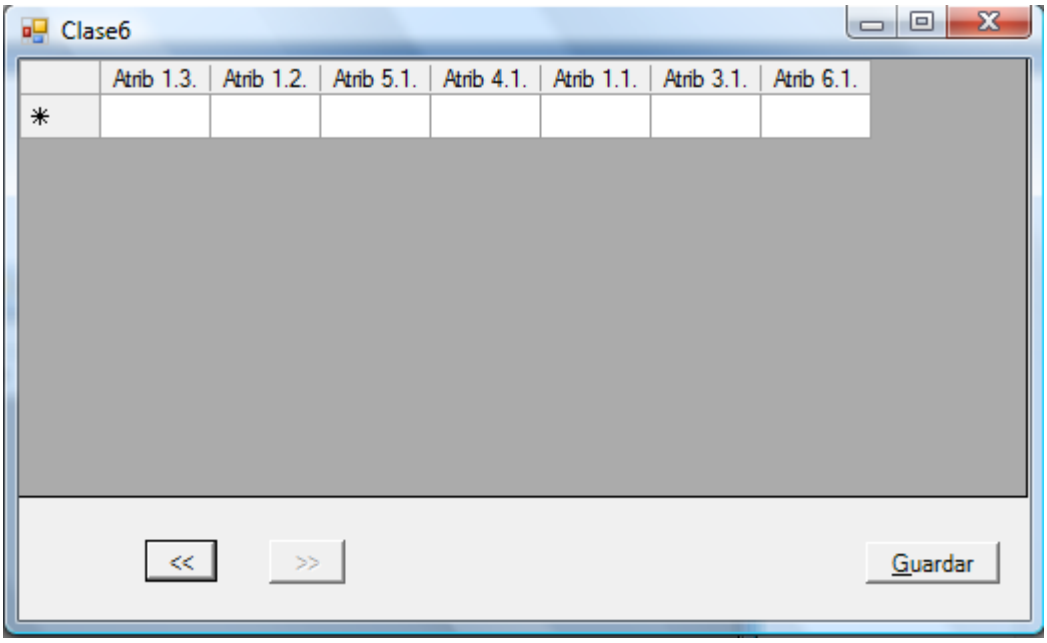


Ilustración 41: Resultado 4.6. Prueba 4

Prueba 5.

El diagrama de elementos dibujado a continuación representa un caso de pruebas, compuesto por seis clases, relacionadas bajo relaciones de herencia y asociación. Se ha de tener en cuenta que la plantilla generada por la Clase3 debe tener además de sus atributos, los atributos de la Clase1, debido a la relación de herencia establecida entre ellas; lo mismo ocurre con la Clase5 con respecto a la clase Clase3. En este caso se han incluido dos detalles que por el momento la aplicación no contempla, una es la relación de agregación y por otro lado la clase asociación (Clase6), ambos aspectos no afectan al resultado obtenido, simplemente no son recogidos en las plantillas.

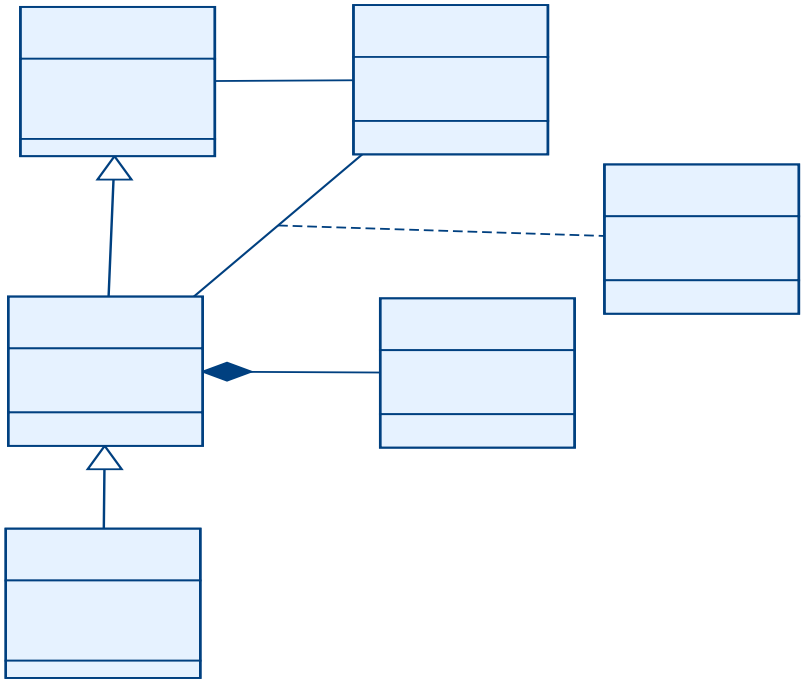


Ilustración 42: Diagrama de clases. Prueba 5

Resultado de la prueba 5.

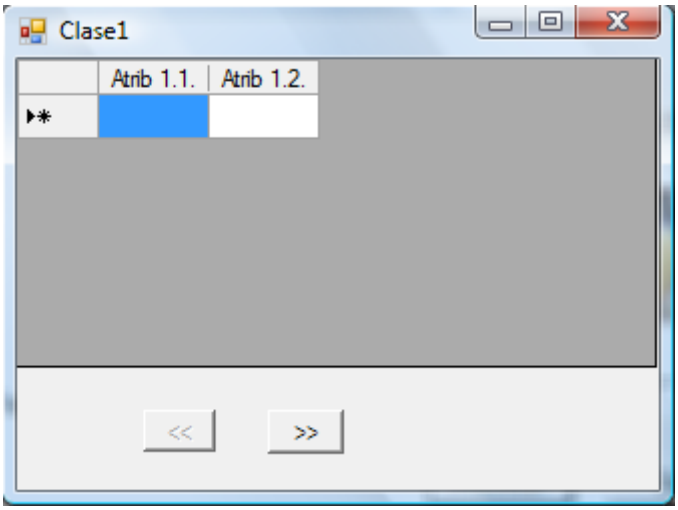


Ilustración 43: Resultado 5.1. Prueba 5

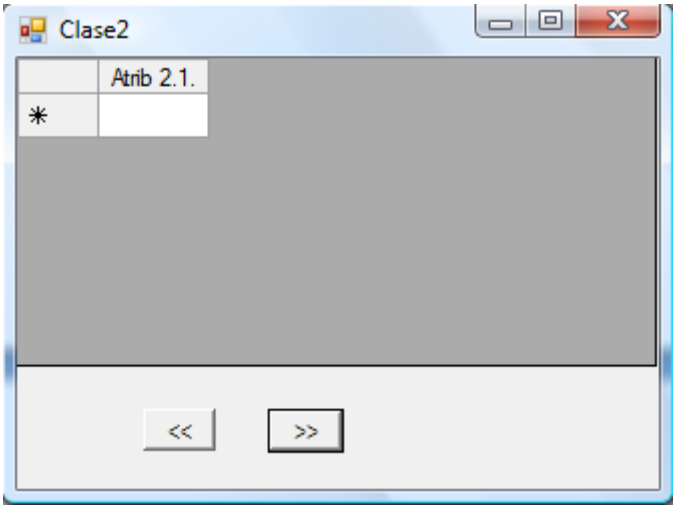


Ilustración 44: Resultado 5.2. Prueba 5

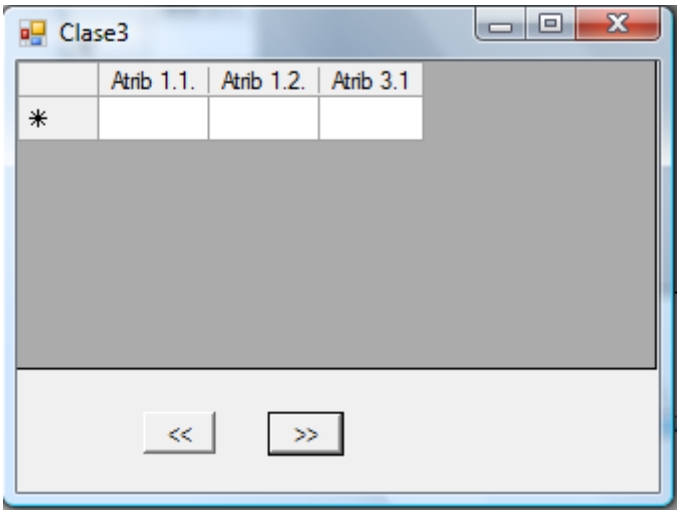


Ilustración 45: Resultado 5.3. Prueba 5

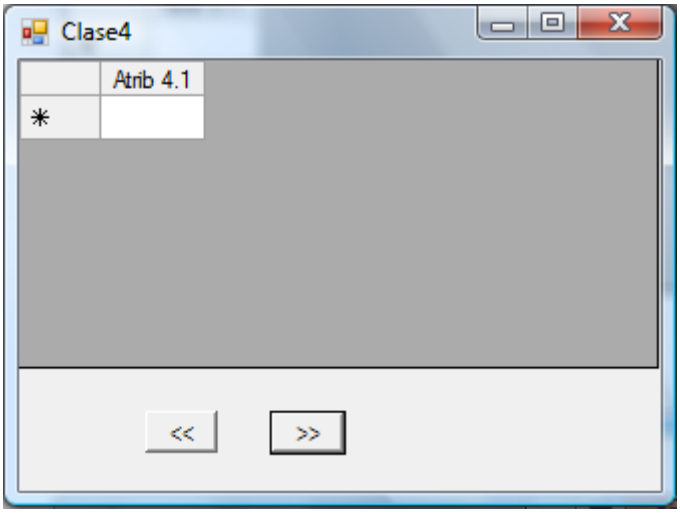


Ilustración 46: Resultado 5.4. Prueba 5

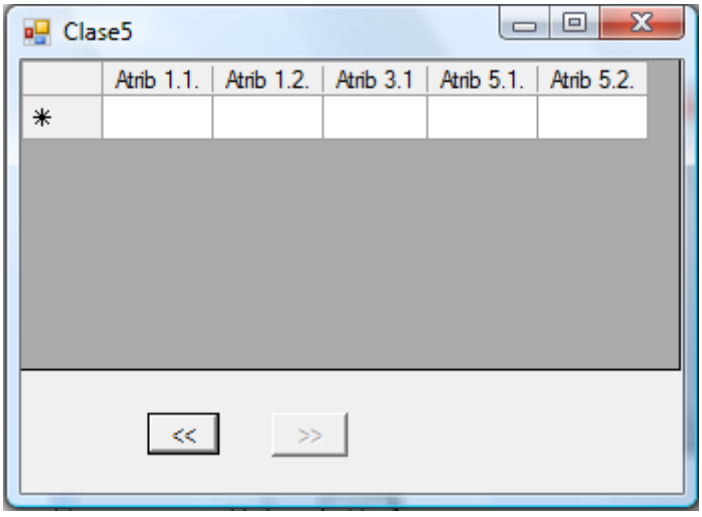


Ilustración 47: Resultado 5.5. Prueba 5

A continuación, se han llevado a cabo una serie de pruebas para corroborar el correcto funcionamiento de la gestión de las plantillas, a partir de los archivos .xml creados a partir de las plantillas de trabajo generadas.

Para chequear la gestión de archivos con formato XML se ha tomado el caso de prueba 4. En la cual se han generado las plantillas de trabajo y se han rellenado con datos distintos, repetidos e incompletos. A continuación, estos datos han sido guardados con formato XML para ser posteriormente recuperados en plantillas observando que no ha habido ninguna modificación y el resultado es correcto.

Por otro lado, se ha llevado a cabo pruebas relacionadas con el funcionamiento de algunos botones y opciones de la interfaz, de este modo las pruebas chequeadas han sido las siguientes:

- Control en la carga de archivos. Es decir, para la carga de archivos solo se permite la entrada de archivos con extensión .csem para las opciones de carga de “Generador de Plantillas” y “Editor de Plantillas” y archivos con extensión .xml para la opción de carga de “Gestionar Plantillas”.
- Control de los botones “anterior” (“<<”) y siguiente (“>>”) dentro de las plantillas. Comprobación de dichos botones, de tal modo que muestren cada una de las clases de diagrama dibujado en las plantillas de trabajo. Controla correctamente el inicio y fin de las plantillas desactivando los botones de “anterior” y “siguiente” respectivamente de la siguiente manera:

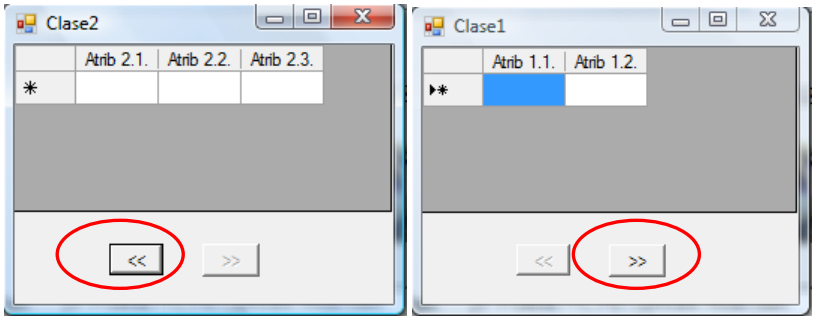


Ilustración 48: Botón Anterior/Siguiente en Plantilla

- Control del botón “Guardar”. Chequeo de dicho botón, comprobando el correcto almacenamiento de los datos bajo el fichero .xml creado por el usuario. Posteriormente se ha comprobado, como se dijo anteriormente la carga de dicho fichero.

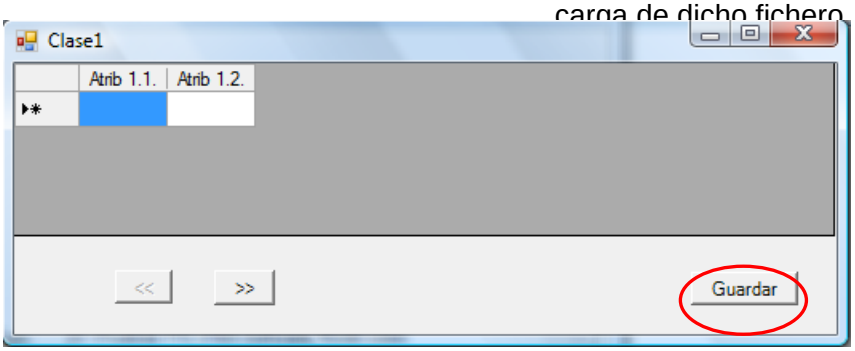


Ilustración 49: Botón Guardar en Plantilla

8 Manual de Usuario

A modo de introducción y para dar a conocer el sentido y principales características de la aplicación desarrollada, se han incluido unas preguntas frecuentes relacionadas con el sistema, que se resuelven a continuación:

- ¿Para qué sirve la aplicación? Dicha aplicación permite la creación, en tiempo de ejecución, de plantillas de trabajo para la gestión de activos software (requisitos, casos de uso, riesgos, pruebas, etc.), estas plantillas serán generadas a partir del diseño de un diagrama de elementos modelado a través de un software específico (SwReuser). El uso de dicha aplicación permite la gestión de patrones de requisitos a partir de plantillas de trabajo.
- ¿Qué posibilidades plantea la aplicación? La aplicación tiene inicialmente tres opciones de trabajo:
    - La generación de plantillas de trabajo vacías a modo de consulta a partir del diagrama de clases creado, en la cual se puede chequear las plantillas generadas a partir de dicho diagrama.
    - La generación de las mismas plantillas de trabajo pero con la opción de ser guardadas para un archivo definido por el usuario.
    - La generación de plantillas de trabajo a partir de archivos con extensión .xml. De este modo, a partir de este tipo de archivos si podemos tener plantillas de trabajo con datos rellenados anteriormente por el usuario.
- ¿Qué es una plantilla de trabajo? Una plantilla de trabajo es una cuadrícula o tabla que representa una clase del diagrama de elementos creado por el usuario. Cada una de las columnas que compone dicha tabla corresponde con los atributos propios de la clase. Y cada una de las filas que puede tener, son instancias insertadas por el usuario.
- ¿Cómo manejar las plantillas de trabajo? El manejo de cada plantilla de trabajo es bastante intuitivo. Para rellenarlo podría ser parecido a una hoja Excel, el uso de los botones no provoca complicaciones, ya que solo tenemos los botones de “Anterior” y “Siguiente” para poder visualizar las plantillas de las distintas clases del diagrama y el botón de “Guardar” para generar el archivo con extensión .xml con toda la información recogida en las plantillas.
- ¿Cómo recuperar las plantillas para futuras ocasiones? Las plantillas guardadas en formato .xml pueden ser recuperadas a partir de la aplicación diseñada, bajo la opción: “Gestionar Plantillas”.

Descripción funcional

En la pantalla principal de la aplicación se observan las distintas opciones claramente diferenciadas:

- Generador de Plantillas.
- Editor de Plantillas.
- Gestionar Plantillas.

Esta pantalla principal del sistema permite al usuario acceder a toda la funcionalidad ofrecida, pudiendo escoger una de las tres opciones indicadas anteriormente. Para ello deberá hacer clic sobre el botón en cuestión.

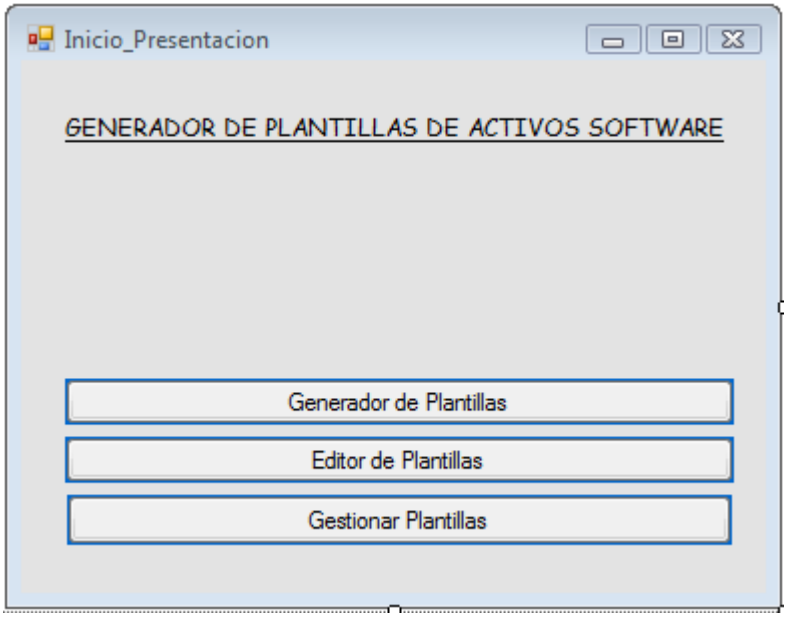


Ilustración 50: Plantilla Inicial

Opción: Generador de Plantillas

Bajo esta opción, el usuario puede visualizar las plantillas de trabajo generadas para cada una de las clases del diagrama de clases, dibujado anteriormente bajo un software ajeno. Estas plantillas de trabajo aunque pueden ser rellenadas, no pueden ser guardadas, ya que su funcionalidad únicamente es brindar al usuario una vista del sistema.

Para poder visualizar las plantillas, antes debemos proceder a la carga de los archivos con extensión .csem que contienen el diagrama de clases, a partir de dicha pantalla:

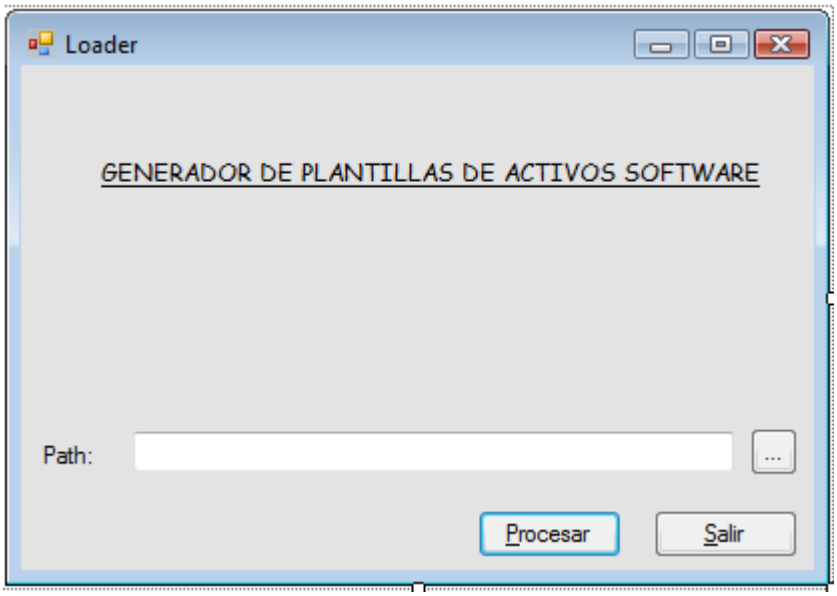
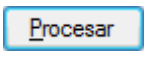


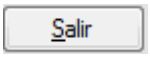
Ilustración 51: Plantilla Cargar archivos .csem

Para cargar un fichero de estas características es necesario conocer los botones de la interfaz indicada:

Este botón permite buscar dicho archivo en las distintas unidades de datos ó dispositivos de almacenamiento.



Este botón, como su nombre indica, permite procesar el archivo .csem buscado generando las plantillas de trabajo.



Este botón permite salir de la aplicación.

Una vez que el archivo ha sido cargado, se podrán visualizar las plantillas de trabajo, de este modo:

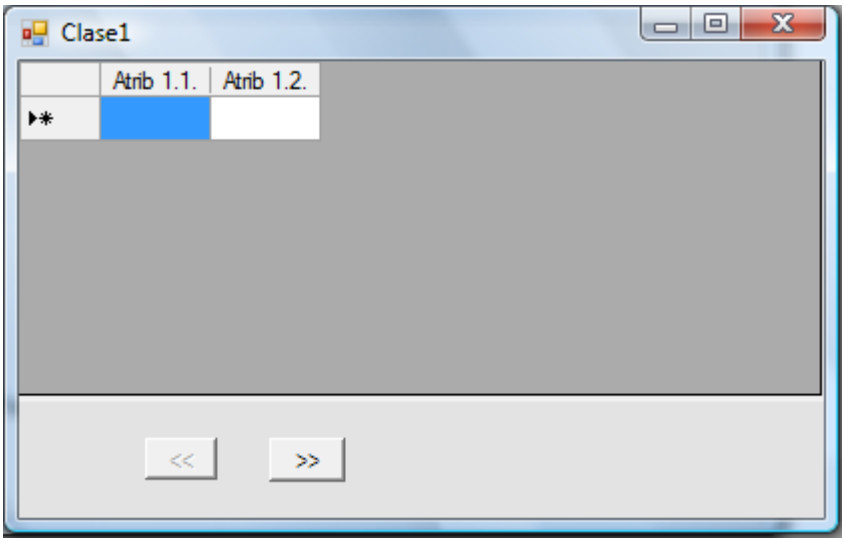


Ilustración 52: Plantilla sin opción a Guardar

Según se puede ver en la plantilla de trabajo, aparece una cuadrícula con tantas columnas como atributos tenga la clase, más los posibles atributos heredados que puede contener. El usuario podrá rellenar dichas plantillas sin ningún tipo de restricción.

Para poder visualizar cada una de las plantillas, es necesario usar los siguientes botones:

Anterior. En el caso de que nos encontremos en la primera plantilla, este botón se encontrará inactivado.

Siguiente. En el caso de que nos encontremos en la última plantilla, este botón se encontrará inactivado.

Opción: Editor de Plantillas

Bajo esta opción, el usuario puede visualizar las plantillas de trabajo generadas, para cada una de las clases del diagrama de clases, como vimos en la opción anterior, pero con la posibilidad de ser guardadas bajo un archivo con formato .xml. En esta opción el usuario puede empezar a generar un histórico de estas plantillas, que es lo realmente interesante de la aplicación.

De este modo, la pantalla utilizada en la carga de archivos .csem es idéntica a la utilizada en la opción anterior, siendo la misma interfaz:

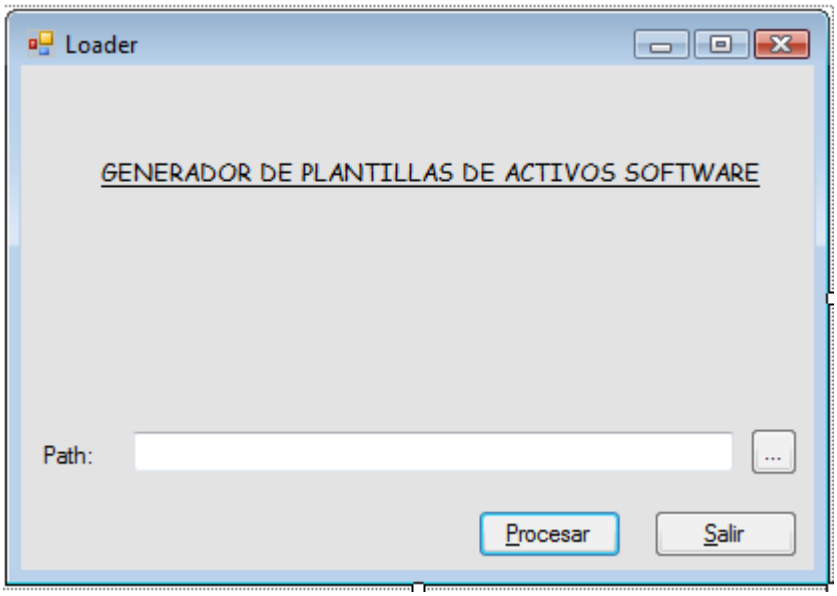


Ilustración 53: Plantilla Cargar archivos .csem

Únicamente, varía la pantalla utilizada para visualizar las plantillas de trabajo, una vez cargadas los archivos necesarios. Esta es la interfaz utilizada en esta opción del menú:

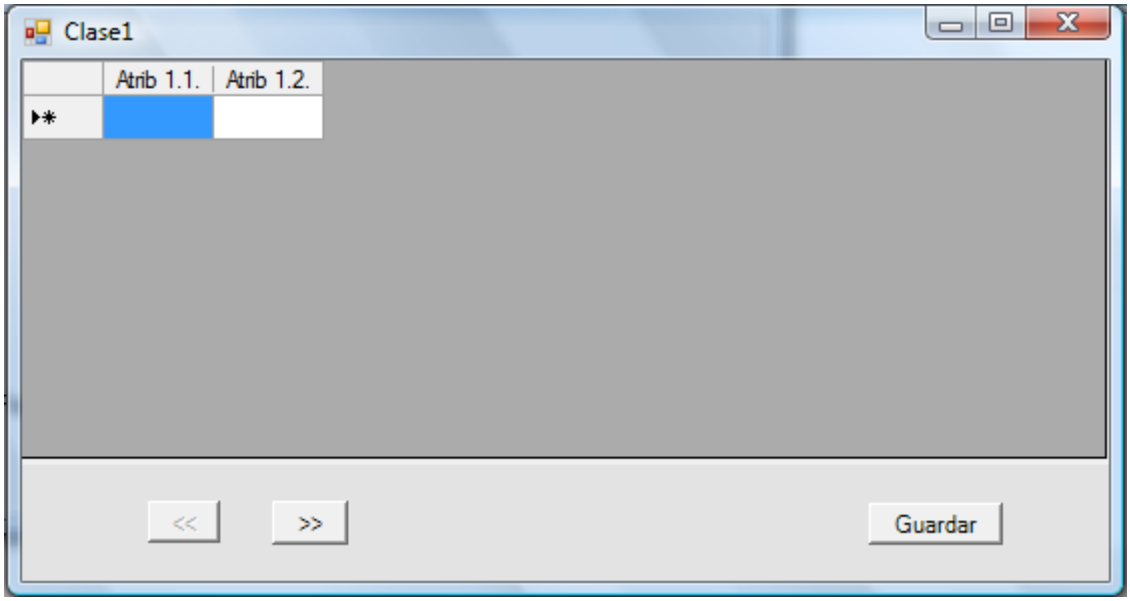


Ilustración 54: Plantilla con opción a Guardar

Como se puede ver, la especial diferencia radica en el siguiente botón 

en el que el usuario podrá guardar las plantillas con sus datos, en cualquier disco o dispositivo bajo un archivo con extensión .xml.

Una vez que las plantillas de trabajo son guardadas correctamente, la aplicación nos avisa de que el proceso ha finalizado correctamente de la siguiente manera:

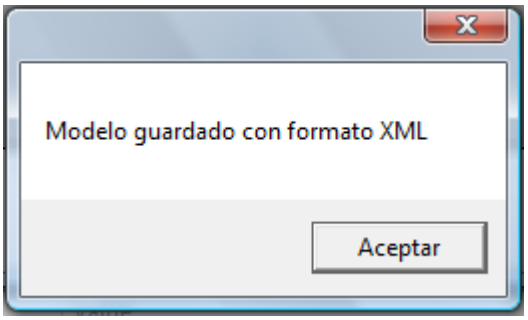


Ilustración 55: Mensaje de aviso. Plantilla guardada

Opción: Gestionar Plantillas

Esta opción, permite cargar archivos con extensión .xml guardados por el usuario anteriormente mediante la opción “Editor de plantillas”. La finalidad de esta funcionalidad del sistema es cargar estos archivos de trabajo y poder tratarlos mediante las mismas plantillas de trabajo.

Para poder visualizar las plantillas, antes debemos proceder a la carga de los archivos con extensión .xml que contienen toda la información necesaria para proceder la carga, dicha carga se realiza a partir de la siguiente pantalla:

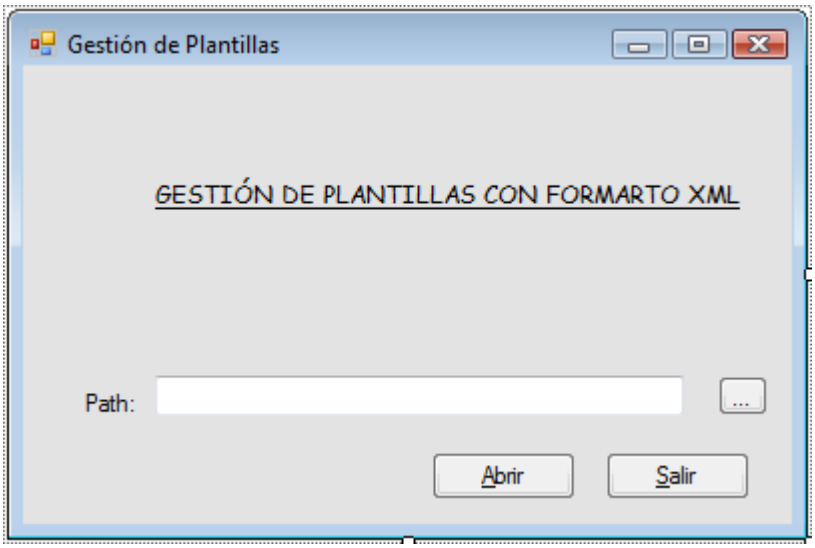
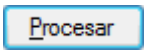
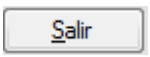


Ilustración 56: Plantilla Cargar archivos .xml

Para cargar un fichero de estas características es necesario conocer los botones de la interfaz:

 Este botón permite buscar dicho archivo en las distintas unidades de datos o dispositivos de almacenamiento.

 Este botón, como su nombre indica, permite procesar el archivo .csem buscado para generar las plantillas de trabajo.

 Este botón permite salir de la aplicación.

Una vez que el archivo ha sido cargado, se podrán visualizar las plantillas de trabajo, en las que tendremos activada la opción de guardar las plantillas, para poder seguir creando históricos:

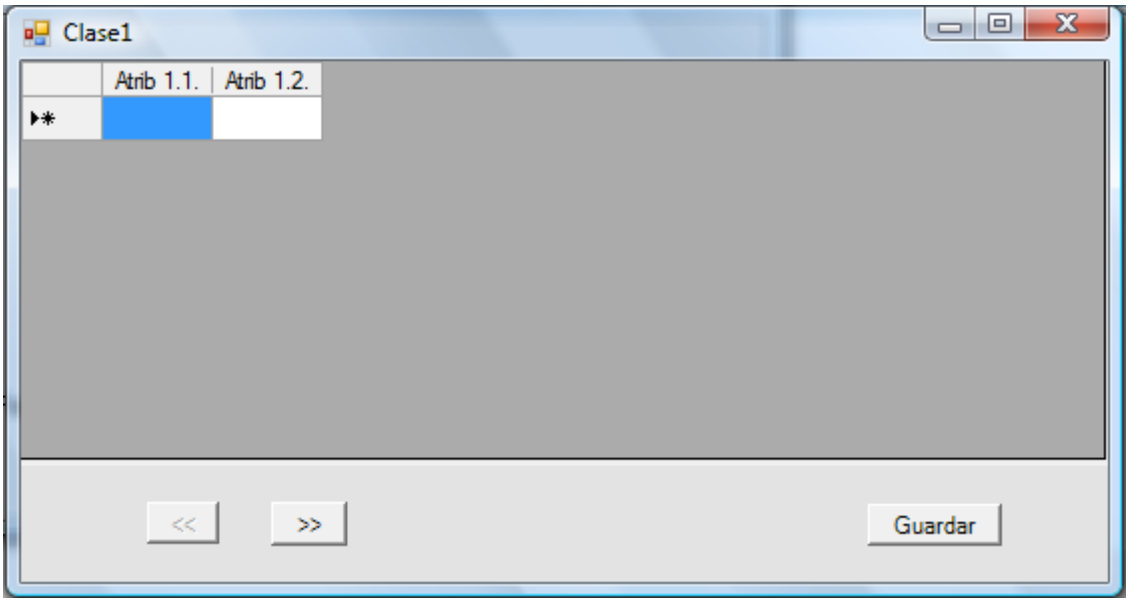


Ilustración 57: Plantilla con opción a guardar

El tratamiento de este tipo de plantilla es el mismo explicado en la opción anterior.

En General el uso de dicha aplicación es bastante sencillo e intuitivo, de hecho se ha diseñado en base a esto, ya que cualquier desarrollo o gestión de información es más eficiente cuando más sencillo es.

9 Conclusiones

Los resultados obtenidos con el desarrollo de este proyecto deben ser calificados de muy satisfactorios, ya que se han conseguido los objetivos más relevantes que se propusieron en un principio.

Se ha conseguido desarrollar un sistema, que proporcione al usuario un seguimiento directo, sobre uno o varios proyectos en función de los aspectos, que inicialmente se consideren importantes por el jefe de proyecto. De este modo, el usuario que trabaja directamente con las plantillas de trabajo generadas por el sistema, puede actualizar la información de estas plantillas, en función de la evolución del proyecto, potenciando la calidad y organización de la información. Esta información, estará disponible para usuarios relacionados con el proyecto, lo cual permitirá una comunicación transversal entre los distintos miembros y facilitará el desarrollo del mismo.

En primer lugar, es importante destacar el hecho de que el modelo de datos en el que se ha pensado recoge parte de la semántica del diagrama de elementos (diagrama de clases) creado por la aplicación SwReuser. De este modo, además de captar las clases y atributos del diagrama de elementos se ha conseguido un paso importante con las posibles jerarquías que se presentan en el diagrama en cuestión dibujado. Así pues, las posibles relaciones de herencia que se establezcan entre los distintos elementos, se reflejan en las plantillas de trabajo utilizadas, sobrescribiendo atributos si fuera necesario. Esta idea, de contemplar la herencia en las plantillas de trabajo no solo enriquece la semántica de sus plantillas sino el trabajo de las mismas. Con todo, como se comentará en el apartado “futuras ampliaciones” la aplicación puede ampliarse para abarcar cada vez más matices del diagrama de elementos, lo que se traduce en la compresión del proyecto en cuestión. El hecho de que las plantillas con las que el usuario trabaje tengan la capacidad de recopilar cada vez más información, contribuye a una mejora en el dominio del proyecto. Lo diseñado y desarrollado por el momento, es una buena base para estas posibles ampliaciones, sobretodo, una vez incorporada la idea de herencia en dichas plantillas.

En segundo lugar, la idea de guardar las plantillas generadas rellenadas o no, en archivos con formato XML fue considerada después de marcar los objetivos del sistema. La idea surgió por propia necesidad de la aplicación, los datos de un proyecto debían ser guardados y considerados para próximas ocasiones, lo contrario, conllevaría más trabajo, inconsistencia en los datos y en definitiva un trabajo inútil. La idea de guardar estas plantillas fue aceptada y muy bien encajada por la aplicación. No supuso grandes problemas en el desarrollo del sistema y permitió una mejora considerable en su uso.

Por otro lado, no sólo se ha intentado encontrar la solución más acertada para el desarrollo del sistema, sino que se ha procurado diseñar una interfaz amigable y fácil de usar. Por este motivo, uno de los objetivos marcados fue conseguir que la navegación entre pantallas fuera muy sencilla e intuitiva para el usuario, pensada para proporcionarle facilidades en el día a día. De esta manera, las plantillas de trabajo con las que el usuario se familiarizará se han insertado en un misma ventana, lo cual está pensado para conseguir una buena legibilidad y manejo de las plantillas, ya que ante proyectos de gran escala, el número de plantillas es bastante grande y si sumamos la dificultad intrínseca que lleva consigo el gestionar proyectos de tal grado con la poca legibilidad que trae consigo el uso de una ventana por cada plantilla, podemos tener problemas de organización en el propio trabajo.

En el diseño y desarrollo de la solución, me he encontrado con una serie de problemas sobre todo relacionados con la falta de conocimiento en las tecnologías utilizadas, sobre todo en Visual Basic .Net. Esto conllevó a utilizar objetos con una serie de limitaciones, como es el caso del objeto ListView, utilizado inicialmente para diseñar las plantillas. En principio todo parecía correcto, hasta que se necesitó tratar la información contenida en cada una de las celdas del ListView, lo cual no fue posible y se necesito un cambio de objeto (Datagridview) que soportara todos los objetivos marcados. Esto trajo consigo un tiempo en poner en orden todo lo hecho hasta el momento y un conocimiento de ambos objetos. Con la tecnología XML no se plantearon grandes problemas, como dije anteriormente considerar la idea de guardar las plantillas de trabajo no fue un conflicto y si otorgó una serie de ventajas a nuestro sistema.

Llegados a este punto, como conclusión personal debo añadir la experiencia enriquecedora obtenida por el hecho de haber trabajado con todas las tecnologías que se han expuesto a lo largo de esta memoria, es especial con Visual Basic .Net y XML, con las que he tomado un primer contacto y he adquirido unos conocimientos bastantes fuertes.

10 Futuras líneas de trabajo

A medida que el proyecto ha ido evolucionando, han ido surgiendo una serie de mejoras que me parece de interés enumerar en este punto del documento, a pesar de haber ofrecido un sistema completo, sencillo de utilizar e intuitivo, hay aspectos que pueden ser mejorables y puntos que se deberían tenerse en cuenta dentro de la aplicación, siempre pensando en las oportunidades de mejora que puede ofrecer la aplicación.

En primer lugar, sería beneficioso iniciar la aplicación con una identificación de usuario. Sería interesante, para delimitar las funciones de los usuarios en la aplicación. De este modo, se asignaría a cada usuario un role específico, que le otorgara una serie de permisos, para acceder a una serie de funcionalidades. En este sentido, se acotaría la implicación de cada usuario en el proyecto, ofreciendo seguridad en la aplicación.

Sería importante, incluir en el modelo diseñado para la aplicación más aspectos de la semántica del diagrama de elementos. Es decir, hasta el momento solo se han tenido en cuenta clases, atributos y relaciones de herencia, pero sería interesante considerar otras relaciones, multiplicidad, tipo de atributos, etc. Teniendo en cuenta toda esta información, el modelo crecería recogiendo más conocimiento del diagrama de elementos dibujado y esto daría lugar a unas plantillas de trabajo cada vez más completas y con más posibilidades.

Por otro lado, sería interesante ampliar la aplicación en el sentido de elaborar informes personalizados, es decir, teniendo en cuenta las fechas de actualización de los datos de las plantillas, obtener opcionalmente informes de seguimiento que permitan al analista y jefe de proyecto observar los puntos alcanzados o aspectos conseguidos.

Tal vez, en un futuro sería beneficioso plantear una arquitectura cliente-servidor, este tipo de arquitectura se basa en que entre todos los ordenadores o dispositivos están en la red, unos ofrecen servicios (los llamados servidores) y otros usan esos servicios (los denominados clientes). En nuestro caso, será la aplicación a través del servidor quien ofrezca unos servicios (las funcionalidades del sistema), y los clientes serán el resto de dispositivos externos que tendrán el acceso remoto a dichas funcionalidades. Este tipo de arquitectura cliente-servidor ofrece la ventaja de un bajo acoplamiento entre las dos componentes que participan.

Cada una de estas mejoras, se pueden tener en cuenta para posibles futuros trabajos, cada una de ellas se compromete en mejorar la aplicación y ofrecer más posibilidades. Tal vez, el camino que se deba seguir para ampliar funcionalidades de la aplicación sea llevar a cabo el segundo punto de este apartado, en el que se habla de completar y ampliar las plantillas de trabajo en función del diagrama de elementos. Por ello, de manera subjetiva, creo que sería el punto más beneficioso a tener en cuenta para conseguir una aplicación funcionalmente completa.

11 Bibliografía

11.1 Portales de Búsqueda

1. Google. (Buscador) **www.google.com**
2. Wikipedia (Enciclopedia on-line) **www.wikipedia.org**

11.2 Páginas Web de Referencia

1. Programación en castellano. **http://www.programacion.net/**
2. Reusercompany. **http://www.reusecompany.com**
3. UML Resource Page **http://www.uml.org/**
4. XML. **http://www.programacion.net/html/xml/principal.htm**

11.3 Libros

1. Erich Gamma. Patrones de Diseño. Addison-Wesley, 2003.
2. Evangelos Petroutsos. La biblia de Visual Basic .NET 2002.
3. [BRAUDE 01] Braude, J. Software Engineering: An Object-Oriented Perspective. John Wiley & Sons, ISBN 0471322083. New York (2001).
4. Stevens, Perdita. Utilización de UML en ingeniería del software con objetos y componentes.
5. Kimmel, Paul D. Manual de UML.
6. Ceballos Sierra, Francisco Javier. Microsoft Visual Basic .NET: curso de programación.
7. Ceballos Sierra, Francisco Javier. Microsoft Visual Basic .NET: lenguaje y aplicaciones.
8. De Boulanger,Thierry. XML Práctico. Bases esenciales, conceptos y casos prácticos.
9. De Livingstone,Dan. XML (Guía Esencial). Pearson educación, 2002.

12 Agradecimientos

Llegamos al momento más reconfortante, pero a la vez el más complicado. No me gustaría dejarme a nadie sin mencionar, ya que son muchos son los que han estado ahí de muchas maneras, prestándome su ayuda en momentos que los necesitaba, desahogándome bajo un café en un momento determinado, o simplemente prestándome sus oído.

En primer lugar, quisiera darle las gracias a mi tutor D. Diego Martín de Andrés por el apoyo y la ayuda que me ha ofrecido a lo largo de este tiempo y sobre todo gracias por todo lo aprendido.

A mis padres y hermanos por su apoyo en los momentos malos, aguantando mi mal humor e intentando apoyarme cuando las cosas no han ido tan bien, gracias por compartir conmigo las alegrías y por los consejos que me habéis dado de los que siempre he podido sacar algo en claro.

A Davinia por ser el teléfono incondicional disponible a cualquier hora, por su cariño y comprensión absoluta. Por ser tú.

A todos mis compañeros y amigos de la universidad, por los buenos momentos que hemos pasado y los que nos quedan, por vuestras ayudas y por saber que siempre os tendré ahí.

A mis amigos, a todos, por vuestras buenas palabras que siempre me han dado fuerza e iniciativa.

A mis animales (Pichi, Chuky y Beni), por vuestro cariño.

A los que no están y a los que nunca pensarían que les estuviera tan agradecida.

En fin, gracias a todos, porque sin todos vosotros esto no habría sido igual.